307-CD-001-003
329-CD-001-003

EOSDIS Core System Project

Flight Operations Segment (FOS) Release Plan and Development Plan for the ECS Project

October 1995

Hughes Information Technology Corporation
Upper Marlboro, Maryland

Flight Operations Segment (FOS) Release Plan and Development Plan for the ECS Project

October 1995

Prepared Under Contract NAS5-60000
CDRL Items 048 and 058

APPROVED BY

Cal E. Moore /s/	9/29/95
Calvin Moore, FOS Segment Manager EOSDIS Core System Project	Date

Hughes Information Technology Corporation
Upper Marlboro, Maryland

307-CD-001-003
329-CD-001-003

This page intentionally left blank.

Preface

The contents of this document define both the development plan and the release plan for the Flight Operations Segment (FOS). Thus, this document addresses the data item descriptions for CDRL 048 and CDRL 058, 307/DV2-001 and 329/DV2-002, respectively.

This document is a formal contract deliverable with an approval code 2. As such, it does not require formal Government approval, however, the Government reserves the right to request changes within 45 days of the initial submittal or any subsequent revision. Changes to this document shall be made by document change notice (DCN) or by complete revision.

Once approved, this document shall be under the Flight Operations Segment Configuration Control Board. Any questions or proposed changes should be addressed to:

Data Management Office
The ECS Project Office
Hughes Information Technology Corporation
1616 McCormick Dr.
Upper Marlboro, MD 20774-5372

This page intentionally left blank.

Change Information Page

List of Effective Pages	
Page Number	**Issue**
iii through viii	Final Copy
1-1 through 1-2	Final Copy
2-1 through 2-2	Final Copy
3-1 through 3-26	Final Copy
4-1 through 4-38	Final Copy
5-1 through 5-2	Final Copy
AB-1 through AB-10	Final Copy
GL-1 through GL-8	Final Copy

Document History

Document Number	Status/Issue	Publication Date	CCR Number
307-CD-001-001 329-CD-001-001	Review Copy	November 1994	94-0175
307-CD-001-002 329-CD-001-002	Final Copy	January 1995	95-0069
307-CD-001-003 329-CD-001-003	Final Copy	October	95-0705

This page intentionally left blank.

Contents

4. FOS Release Development Plan

5. FOS Development Schedules

Tables

Abbreviations and Acronyms

Glossary

1. Introduction

1.1 Identification

This document is the FOS Release Plan and Development Plan for the ECS project which are items 048 and 058 on the Contract Data Requirement List (CDRL) and defined by Data Item Descriptions (DIDs) 307/V2 and 329/DV2 under contract NAS5-6000.

1.2 Scope

The development plan component provides detailed plans of technical development factors required to implement the FOS. The plan identifies a phased implementation approach for the development of the FOS, and allocates segment functions among the phases. The plan shows the organizations of the development effort to the lowest level of the Work Breakdown Structure (WBS). Technical efforts and schedules associated with the development are consistent with overall segment development plans.

The release plan component includes a reference to the schedules for the segment builds and maps the builds into releases. This plan identifies the implementation approach for the development of the FOS, including the partitioning of the task into release and builds within each release.

This document is under the FOS Configuration Control Board (CCB) for the November 1994 draft submittal and the final Release A submittal. Changes to these volumes must be approved by this CCB prior to inclusion in this document.

This document reflects the August 23, 1995 Technical Baseline maintained by the contractor configuration control board in accordance with ECS Technical Direction No. 11 dated December 6, 1994. It covers releases A and B for FOS.

1.3 Purpose

This document defines the Flight Operations Segment (FOS) developmental release plan. It focuses on delineating the approach that will be taken to incrementally develop the FOS.

1.4 Status and Schedule

This submittal of DIDs 307/DV2 and 329/DV2 meets the milestone specified in the Contract Data Requirements List (CDRL) of NASA contract NAS5-60000. It is anticipated that this submittal will be reviewed during the appropriate segment- or system-level post Preliminary Design Review (PDR) timeframe, and that subsequent changes to the document will be incorporated into a resubmittal according to a schedule mutually agreed to by GSFC and ECS. This document is a final version.

1.5 Document Organization

The document is organized to describe the approach to develop the FOS:

Section 1.0 provides information concerning the identification, scope, and organization of this document.

Section 2.0 provides a list of applicable documents, which were used directly or indirectly in the preparation of this document.

Section 3.0 defines the FOS-unique development guidelines and standards. These development guidelines and standards are based on the ECS Software Development Plan, ECS System Engineering Plan, and the ECS System Implementation Plan. In general, this section will reference these three documents. Any unique additions or clarifications pertaining to the development guidelines and standards in relation to these three project plans will be documented in this section.

Section 4.0 defines the FOS release development plan. This includes the allocation of functions to releases. For FOS, this allocation is between Release A and Release B. The defined functions or threads in the system will be organized to show the phasing of the development effort within a release. In particular, the dependencies and sequencing of the development effort will be outlined in this section.

Section 5.0 references the FOS development schedules. It will build on the information included in section 4.0, and define the dependencies both internal to FOS from the ECS project, and external dependencies (e.g., spacecraft and instrument dependencies). Note: the specific FOS development schedules will be referenced in this section, not explicitly included.

Abbreviations and acronyms contains an alphabetized list of the definitions for abbreviations and acronyms used with this document

Glossary contains the key terms that are included with this release plan.

2. Related Documentation

2.1 Parent Document

The parent documents are the documents from which this FOS Release Plan's scope and content are derived.

308-CD-001-004	Software Development Plan for the ECS Project

2.2 Applicable Documents

The following documents are referenced within this document, or are directly applicable, or contain policies or other directive matters that are binding upon the content of this volume.

194-201-SE1-001	Systems Engineering Plan for the ECS Project
301-CD-002-003	System Implementation Plan for the ECS Project
401-CD-001-002	Verification Specification for the ECS Project

2.3 Information Documents

The following documents are referenced herein and, amplify or clarify the information presented in this document. These documents are not binding on the content of the FOS Release Plan.

194-102-MG1-001	Configuration Management Plan for the ECS Project
194-202-SE1-001	Standards and Procedures for the ECS Project
193-208-SE1-001	Methodology for Definition of External Interfaces for the ECS Project
194-317-DV1-001	Prototyping and Studies Plan for the ECS Project
194-401-VE1-002	Verification Plan for the ECS Project, Final
194-415-VE1-002	Acceptance Testing Management Plan for the ECS Project, Final
194-501-PA1-001	Performance Assurance Implementation Plan for the ECS Project
194-502-PA1-001	Contractor's Practices & Procedures Referenced in the PAIP for the ECS Project
604-CD-001-004	Operations Concept for the ECS Project: Part 1-- ECS Overview
604-CD-002-001	Operations Concept for the ECS project: Part 2B -- ECS Release B, Annotated Outline
604-CD-003-001	ECS Operations Concept for the ECS Project: Part 2A -- ECS Release A, Final

194-00285TPW	Technical Paper: A Glossary for the ECS Project
222-TP-003-006	Release Plan Content Description for the ECS Project, Technical Paper
423-16-01	Goddard Space Flight Center, EOSDIS Core System Project, Data Production Software and Science Computing Facility (SCF) Standards and Guidelines
NHB 2410.9A	NASA Security Office, Security, Logistics and Industrial Relations Division; NASA Automated Information Security Handbook
MIL-HDBK-217F	Military Handbook: Reliability Prediction of Electronic Equipment
MIL-HDBK-472	Military Handbook: Maintainability Predictions

3. FOS Development Guidelines and Standards

3.1 Background

This section defines the FOS-unique development guidelines and standards. These development guidelines are based on the ECS Software Development Plan, ECS System Engineering Plan, and the ECS System Implementation Plan. Any unique additions or classifications pertaining to the development guidelines and standards are included in this section.

Included are lists of engineering products available upon completion of the preliminary and detailed design phases, guidelines for object oriented development during preliminary and critical design, standards for documenting the dynamic models and interface class descriptions during design phases, and the segment configuration management approach. FOS prototype development is performed in accordance with section 4.1.6 of the ECS Software Development Plan. The following table represents the set of FOS software development documentation, which will be delivered throughout the development life cycle.

Table 3-1. FOS CDRL Summary Table (1 of 2)

Document	DID	Relative Due Date	Notes
PDR/IDR			
FOS Requirements Specification	304/DV1	2 wks before PDR/IDR	
FOS Design Specification	305/DV3	2 wks before PDR/IDR	
FOS Release Plan	307/DV2	2 wks before PDR/IDR	
FOS Integration and Test Plan	319/DV1	2 wks before PDR/IDR	
FOS Development Plans	329/DV2	2 wks before PDR/IDR	Included in 307
FOS Database Design & Schema Specification	311/DV1	1 mo after PDR/IDR	Included in 305
Spacecraft Analysis Interface Control Document (P)	209/DV	2 wks before PDR/IDR	
Data Format Control Document(P)	209/SE2	2 wks before PDR/IDR	
ASTER Interface Control Document (P)	209/SE2	2 wks before PDR/IDR	
FOS Service Internal Interface Control Document (P)	313/DV	2 wks before PDR/IDR	
CDR			
FOS Design Specification	305/DV2	2 wks before CDR	
FOS Operations Scenarios	605/OP2	2 wks before CDR	
Spacecraft Analysis Interface Control Document (C)	209/SE2	2 wks before CDR	

Table 3-1. FOS CDRL Summary Table (2 of 2)

Document	DID	Relative Due Date	Notes
Data Format Control Document(C)	209/SE2	2 wks before CDR	
ASTER Interface Control Document (C)	209/SE2	2 wks before CDR	
FOS Service Internal Interface Control Document(C)	313/DV	2 wks before CDR	
Science Data Verification Facility Interface Control Document	209/SE2	2 wks before CDR	
FOS Release Plan	307/DV2	2 wks before CDR	
FOS Development Plan	329/DV2	2 wks before CDR	Included in 307
Other			
Software Critical Items List	520/PA2	2 wks before I&TR ATR	
FOS Integration & Test Procedures	322/DV3	TRR	
Maintenance & Operations Procedures	609/OP1	2 mo before release	
FOS Integration and Test Reports	324/DV3	2 wks before CSR (PR) 6 wks before RRR (F)	
ECS Operations Plan	608/OP1	1 mo before RRR	
Programmer's Manual	612/OP3	1 mo before release	
Discrepancy Reports	413/VE3	2 mo after release	
Software Nonconformance Reports (formal)	521/PA3	monthly/starting with I&T activities	
Prototyping and Studies Final Report	331/DV3	6 mo before K end	

(P) = PDR (C) =CDR K= Contract (Pr) = Preliminary (F) = Final

3.2 Engineering Products

The engineering products section identifies the output of the preliminary and detailed design phases. Section 3.2.1 lists those products associated with the preliminary design phase and section 3.2.2 lists those associated with the detailed design phase.

3.2.1 Preliminary Design Phase Engineering Products

The following is a list of engineering products produced during the preliminary design phase by the FOS team:

L4 Requirements Specification

The FOS Requirements Specification includes:

Level 4 Requirements

- Definition of the Level 4 requirements including general (Vol. 1) and mission-specific (Vol. 2) requirements

Level 4 Requirements Traceability

- A trace of Level 4 requirements to: Level 3 requirements, release, test method (test, demonstration, inspection, or analysis), IST requirement (yes or no), CSMS requirements (marked as 'X' if requirement is fulfilled by CSMS services)
- A trace of interface requirements from the Interface Requirements Document(s) (IRDs) to Level 4 requirements

Design Specification

The FOS Design Specification includes a description of the FOS architecture including both hardware and software. Additionally, the FOS Design Specification includes the following items for each FOS subsystem:

Preliminary Context Diagram

- A context diagram for each FOS subsystem that shows all interfaces with other subsystems and external entities

Preliminary Object Models

- Object models provide multiple views that focus on different portions of the subsystem at various levels of abstraction
- Every class is fully expanded in at least one view
- All attributes and operations that play a significant role in meeting a requirement are represented
- All interface classes are defined

Preliminary Dynamic Models

- Dynamic models include a set of scenarios and event trace diagrams that are representative of nominal processing conditions

Preliminary Functional Models (Optional)

- Functional models contain descriptions and data flow diagrams of complex operations found within the object model

Performance Description

- Textual description of performance requirements applicable to a particular subsystem and an approach to fulfilling these requirements.

Data Dictionary

- A description of each class and it’s associated attributes and operations

Level 4 Requirements Trace to Classes

- Traceability of Level 4 requirements to classes

Integration and Test Plan

The FOS Integration and Test Plan includes the following items:

Test Case Definition

- Test cases are based on segment scenarios

L4 Requirements Trace to Scenarios

- Maps the segment scenarios to level-4 requirements

3.2.2 Detailed Design Phase Engineering Products

The following is a list of engineering products produced during the detailed design phase by the FOS team:

Design Specification

The Design Specification produced during the critical design phase contains a level of detail which is sufficient to allow the system to be coded. This includes PDL for all nontrivial operations. Because the dynamic models and the functional models are translated into object model constructs, the object models provide a complete map to the implementation. The detailed dynamic models and functional models are included to enhance the understanding of the object models.

The FOS Design Specification includes a description of the FOS architecture including both hardware and software. Segment level event traces are provided for key segment scenarios. Additionally, the FOS Design Specification includes the following items for each FOS subsystem:

Updated Context Diagram

- Context diagram, for each FOS subsystem, that shows all interfaces with other subsystems and external entities
- Reflects modification to the Preliminary Context Diagram as a result of the detailed interface definition process

Detailed Object Models

- Object models provide multiple views that focus on different portions of the subsystem at various levels of abstraction
- Object models are partitioned into tasks
- Every class is fully expanded in at least one view
- All attributes and operations needed for implementation are represented with signatures provided for each
- All interface classes are defined and the owner subsystem identified

Detailed Dynamic Models

- Dynamic models include a set of scenarios and event trace diagrams that are representative of boundary and erroneous processing conditions as well as the nominal processing conditions
- State diagrams are included as necessary for non-trivial interface classes to identify additional attributes, operations and associations necessary for the implementation of the interface
- State diagrams are included as necessary to define complex interactions between classes

Detailed Functional Models (Optional)

- Functional models contain descriptions and data flow diagram of all complex operations identified within the object model

Performance Description

- Textual description of performance requirements applicable to a particular subsystem and an approach to fulfilling these requirements.

Data Dictionary

- A description of each class and it's associated attributes and operations

Level 4 Requirements Trace to Classes

- Traceability of Level 4 requirements to classes, and if applicable, operations within a class

Program Design Language (PDL)

- PDL is provided for all non-trivial operations defined for each class
- The FOS PDL follows the C++ syntax with class definitions, member functions, and comments (NOTE: this is a deviation from the ECS Software Development Plan which defines a FORTRAN-like syntax for PDL)

Unit Test Plans

- Unit test plans/cases are provided in conjunction with unit code inspection. (Note: this is a deviation from the ECS Software Development Plan which specifies that unit test plans/cases are provided during design walkthroughs.)

Interface Definition

- Definitions for all interfaces between FOS subsystems, as well as interfaces with external entities are provided
- Interface definitions include the source, destination, class(es), data structures, nominal frequency, and description

Database Definition and Schema Specification

- Physical representation of the data base schema to be implemented in support of FOS subsystem operations

3.3 Object Oriented Development Guidelines

This section provides guidelines for the object oriented development approach utilized during preliminary and detailed design. The descriptions of object oriented design principles serve as tutorials to support development of those criteria which either support or are included in the list of engineering products to be produced during the design phases.

3.3.1 Preliminary Design Criteria

FOS preliminary design is preceded by the system design phase. The products of the system design phase are the inputs into preliminary design.

3.3.1.1 Preliminary Design Entrance Criteria

Software preliminary design establishes the software architecture necessary to satisfy the requirements by transforming the analysis model into a design model. This transformation results in the restructuring of the model to reflect components, their interfaces, and the outlines of a subset of classes that are significant to the architecture. Deciding how to decompose the subsystems into components is an important step of preliminary design. The subsystems are decomposed recursively into smaller and smaller components until they are small enough to design directly from primitive or library elements. Once a satisfactory partitioning is achieved, a preliminary description of the interfaces between the components is able to be defined. In addition, a general strategy for implementing associations may be identified. Design of external interfaces is coordinated with appropriate representatives from other subsystems. The classes that are important to the architecture, or that are needed to satisfy important requirements, are populated with operations and attributes. Next, the preliminary design is optimized, with a focus on optimizing the overall software architecture. Finally, requirements mappings are updated to account for model changes that occur during preliminary design.

3.3.1.1.1 Updated Scenarios Defining Subsystem-level Interactions

A set of segment scenarios is established when the Preliminary Design Phase is entered. At this point the scenarios reflect the steady state behavior of the system.

- Reflect Operations Concept Document

  Scenarios defined during the preliminary design are derived from the ECS Operations Concepts Document. The Operations Concept Document captures the system's mission, its phases and functions, constraints, configurations, and external interfaces. It reflects the requirements in various situations, both normal and anomalous.

- Reflect Level 3 FOS Requirements

  Scenarios defined during the preliminary design we derived from the ECS Functional and Performance Requirements Specification. This document reflects the Level 3 requirements for the ECS including functional, performance and interface requirements.

3.3.1.1.2 Requirements Trace Matrix

Requirements traceability matrices are created, mapping Level-3 requirements to subsystems and classes.

- L3 to Subsystems and Classes

  The L3 requirements are allocated to specific subsystems and are also mapped to specific model entities within those subsystems. At a minimum, requirements are mapped to classes and associations found on the object model. If necessary for clarification, mapping is made to specific attributes and operations. As requirements are decomposed, their representations in the models are refined, which causes the mappings to become more specific.

3.3.1.1.3 Subsystem Object (static) Model

An analysis model is developed to identify the classes and associations from the application domain that are directly relevant to the problem at hand. It is unnecessary to define all attributes for every class, to make a final distinction between aggregations and associations, or to define the correct multiplicity for every association at this time. These kinds of issues are refinements to the basic static structure of the system and are therefore of secondary importance. Prior to entering the Preliminary Design Phase, the original analysis model is updated to reflect segment architecture design decisions (see Section 3.3.1.1.6 below).

3.3.1.1.4 Subsystem Event Traces

All scenarios are examined for events that are external to the segment: e.g., signals, inputs, decisions, interrupts, and actions to/from external actors (hardware, software, users). Events that are exchanged between classes within the segment also are identified. The sender, receiver, and attributes of an event are documented. The events may also be organized into a generalization hierarchy so that they can be parameterized. The sender and receiver of each event should have been documented. The scenarios are converted into event trace diagrams, that are representative of nominal processing conditions.

3.3.1.1.5 Preliminary Data Dictionary

At this point the data dictionary contains the names and descriptions of the classes. A check is be made to ensure that class names are unique.

3.3.1.1.6 Segment Architecture

During System Design phase, the analysis model should have been modified as needed to reflect segment level architectural decisions. The following bullets summarize some of the main architectural decisions that are made during this phase.

- Set priorities to guide trade-offs during the rest of the design.

- Partition segment into subsystems (can be multi-tiered) on the basis of common properties (cohesiveness of function, physical location, phase of execution, etc.) down to subsystem level, and allocate subsystems to software, hardware, or human operations.
- Look at segment threads to see if a number of them interact with a single subsystem. This may indicate a segment level data repository. Establish the data management mechanism for each such store, perhaps selecting a segment-wide common mechanism.
- Examine segment threads to identify segment resources and establish control mechanisms. Determine which segment threads are initiated or terminated by an external user and determine which subsystems are involved. Modify the models and event routing accordingly. Establish a plan for user interfaces: look and feel, interface standard, etc.

3.3.1.2 Preliminary Design Review Completeness Criteria

3.3.1.2.1 Preliminary Object Models

In the object models, all classes are identified, implying that all requirements are mapped to a specific class or classes, not that all classes included in the final detailed design are identified. Note: during detailed design, the models will continue to be refined for the purposes of reuse, optimization, encapsulation of interfaces, etc.

The object model is examined to ensure that it is really an object model and not a data flow diagram disguised as an object model. There are certain characteristics that are flags. First is the nature of the instances of the classes that make up the diagram. Object-oriented designs are usually made up of classes for which there are many instances of each. If the diagram is made up of classes which, by and large, have just one instance, then the classes are probably just functions with object wrappers. Another flag is that the names of the associations connote the passing of data to another phase of processing. A third indication is that the names of the classes generally imply functionality rather than things (Manager, Controller, Processor, etc.). Nominally there is one controller class per object model.

Every allocated requirement is mapped, to the class level. Some requirements may require collaboration of classes for coverage, which means that associations must be identified as well.

Class Definitions

The attributes that are identified during preliminary design are those that are visible to other classes in the segment. The internal details of the classes are not included during preliminary design. If attribute data types are known then they are included, as well as known constraints. Derived attributes, unless carried over from the analysis model, are not included.

The operations that are identified during preliminary design are those that are intended to be invoked by other classes. Therefore, sub-operations are not generally placed in the preliminary design model. Implied operations such as attribute and association access operations, constructors, and destructors are not included in the preliminary design phase.

All operations support a requirement and no superfluous operations are included. During detailed design, additional operations may be added -- to improve reusability of a class, for example. Each

operation represents a distinct behavior that is part of the definition of the class to which the operation belongs. The overall behavior of a class is represented by the full set of its public operations.

Interface Classes

Interface classes have the minimum functionality needed to affect the interface. The interface class is not responsible for controlling the internal behavior of the subsystem. Instead, the interface class is associated with a controller class, if necessary. The complexity of the interface class depends upon the complexity of the data being passed between the subsystems and the complexity of the behavior of the other subsystem. Interface class complexity can be lowered and its resistance to change enhanced through the use of protocol classes. Communications protocol classes have the responsibility of passing data from one subsystem to another over a network. An inter-subsystem interface class may require the behavior of a communication protocol to help it interface between the two subsystems. The inter-subsystem interface class handles the high level aspects of the interface, and the communications protocol class handles the low level details of passing data from one side of the interface to the other.

An inter-subsystem interface encapsulates the state and behavior of another subsystem. Within one subsystem it is representative of the other subsystem. All communications to the other subsystem goes through the inter-subsystem interface. When there are many inter-subsystem interfaces, it is sometimes desirable for interface management reasons to use the same interface class on both sides of the interface.

3.3.1.2.2 Preliminary Dynamic Models

For each subsystem, the set of segment scenarios is examined to identify those scenarios that involve the subsystem. This is done by looking at the segment threads and identifying each one that flows through the subsystem.

A set of event traces is created for each subsystem. Any entities (classes, attributes, associations, etc.) that are discovered are added to the models. For each event defined in the event traces, a sender and a receiver are identified. This provides a basis for coordinating the negotiation of interfaces between segments and between subsystems.

For each subsystem, all external interfaces are identified and captured. Interfaces are encapsulated within boundary (interface) classes within the subsystem. The attributes and operations of the interface classes are defined such that all critical data (attributes, operations, or signals) to pass the subsystem boundary are identified as are the operations needed to effect the transfers. This does not mean that the full data representations or methods are to be defined in the PDR time frame.

3.3.1.2.3 Overall Consistency

General overall consistency will be a by-product of following the preceding outline in the development of the preliminary design model.

Level 4 requirements map to Level 3 requirements. Level 4 requirements map to classes and dynamic models map to classes. This provides an unbroken chain by which high level

requirements are broken down in to requirements of ever increasing detail, while the corresponding design models are refined and mapped back to the requirements (i.e. requirements are mapped to classes).

Inspections can only be completed with a complete data dictionary. Therefore, a data dictionary is prepared with entries for every modeling entity, It contains a clear description of each class and any assumptions on its membership or use. The data dictionary also describes associations, attributes, and operations.

3.3.2 Detailed Design Criteria

3.3.2.1 Detailed Design Entrance Criteria

The products of the preliminary design phase are the inputs into the detailed design phase. Reference Section 3.2.1 of this document for the list of products and brief descriptions.

3.3.2.2 Detailed Design Phase Completeness Criteria

Detailed design is the process of identifying and designing all software classes that comprise each component. The purpose of detailed design is to establish a stable, well defined detailed design that is based upon the requirements allocations and software architecture established during the preliminary design phase. A detailed design model is created to aid in the definition of the design.

The software detailed design model is a further refinement and expansion of the preliminary design model. During preliminary design, operations are mapped to the object model from the dynamic and functional models only for the architecturally significant classes. During detailed design, this mapping is done for the rest of the classes, and the state diagrams of the dynamic model are mapped to specific implementations. The segment event flow diagram establishes the dynamic context of each of the subsystems in the segment.

By CDR, all diagrams are complete and consistent. The detailed design phase completeness criteria are discussed below.

3.3.2.2.1 Detailed Object Models

During detailed design the control scheme reflected in the dynamic model must be translated into an implementation. This process is discussed later when the dynamic model is addressed.

With the basic building blocks of the design in place, optimization of the object model is performed. The existing representations are transferred, and details added, to support efficient information access and use . The focal point is the object model, but all models are affected. Optimization of the object model introduces efficiency into the software data structures and algorithms. Several techniques are applied, as follows:

- Derived entities are added
- Derived entities, including attributes, operations, and classes, are introduced to establish direct access paths and persistence of data values. Updating derived entities can cause

update anomalies. They may also introduce new dynamics to be modeled. A trade off between implementation cost and efficiency gained is determined.

- Inheritance is verified and maximized
- The object model is analyzed and adjustments are made to maximize inheritance. Adjustments include modifying argument lists, moving attributes and operations from a subclass to a superclass, and abstracting common behavior from several classes to form a superclass. As the level of nesting approaches ten, however, the complexity introduced to the model outweighs the benefits of inheritance.
- Associations are designed
- During preliminary design a default implementation is selected for each kind of association. During detailed design, each association is re-examined on the basis of multiplicity, traversal characteristics, access characteristics, etc., to decide whether to implement that specific association differently. It may be necessary to modify the object model -- usually the attributes and the access operations or methods -- to reflect the changes.
- Coherence of entities are verified

  All classes and operations on the object model are reviewed to ensure that the parts work together toward a common goal. One indication that a class is not coherent is the number of attributes, operations, and associations it contains. As a guideline, a class that has more than ten attributes, more than ten associations, or more than twenty operations is divided.
- Data structures are refined
- Existing data structures are reviewed to determine modifications that might increase algorithmic efficiency.
- By the end of detailed design, there are PDL representations for all nontrivial operations or methods.

Class Definitions

The development of the inter-subsystem interfaces is closely coordinated between the subsystems. During detailed design, the data content and identification of the protocol for all inter-subsystem interactions is documented. State diagrams are developed, as applicable, for classes with significant dynamic behavior. Classes that handle external events are considered to fall into that category. State diagrams are developed for inter-subsystem interface class, as applicable. It is the responsibility of the owning subsystem to develop the state diagram. The following criteria are used for defining final representations of all classes:

- Class Satisfies Class Inspection Criteria

  The class satisfies the Detailed Object Design Inspection checklist defined in the ECS Software Inspection Process.
- Define All Attributes

The attributes providing the internal details of the classes are included. Attribute data types, default values (if any) and constraints (if any) are shown. Derived attributes and their full signatures also are included.

- Identify All Operations

  Although it is not necessary to show all operations on the object models, all operations that are intended to be implemented are defined. This includes access operations. An access operation is one that reads or writes and attribute or association of an object or class. A read and a write operation is defined for each attribute in each class, including instance attributes, class attributes, and derived attributes. A read and a write operation is defined for each association. Signature definitions of each operation are included in the object model and data dictionary. The algorithms for update operations on derived attributes are written. The access privilege is determined for each operation and the object model is annotated. The access privilege specifies the scope of visibility of the particular feature.

- Operations Perform One Function
- Operations Size is Reasonable

  As a guideline the average operations size is less than 100 LOC, nominally around 50 LOC.

Interface Classes

For each component, all external interfaces, interfaces to other subsystems, and interfaces to other components are identified and captured. Interfaces are encapsulated within boundary (interface) classes within the component. The attributes and operations of the interface classes are defined such that all data (attributes, operations, or signals) to pass the component boundary are identified, as are the operations needed to effect the transfers. This includes definitions of the full data representations and methods.

Interface classes that are shared by subsystems must be clearly assigned to one or the other subsystem for development. The class will appear in the models of both subsystems, but any changes to the class must be done by the subsystem that is responsible for the class.

Reference the discussion on Interface Classes in section 3.3.1.2.1 for more detailed information describing interface classes.

3.3.2.2.2 Detailed Dynamic Models

Scenarios

In the detailed design phase, all significant subsystem scenarios in which component's participate are represented in the final dynamic model. For each component, the complete set of subsystem scenarios is examined to identify those scenarios that involve the component. This is done by looking at the subsystem threads and identifying each one that flows through the component.

Scenarios are developed to reflect all aspects of the use of each class, including error conditions. It is likely that additional events, attributes, and even objects will be discovered. These scenarios

may reveal complex dynamic behavior that was not previously recognized, thus necessitating the development of additional state diagrams.

A sequence of classes represents the scenario/thread. An event trace diagram is created for component scenarios, as applicable. Event trace diagrams are examined to identify classes with significant dynamic behavior. By examining all event traces, classes with significant dynamic behavior are identified and are modeled via state diagrams, if applicable. Any entities (classes, attributes, associations, etc.) that are found to be needed are added to the models.

State Diagrams

State diagrams are pertinent for those classes that have state. This means that an object of the class can react differently to the same event depending on what has occurred to the object previously. Usually, this amounts to only a relatively small percentage of the classes in the model.

State diagrams, when used, are reviewed for completeness and accuracy. Verification is performed to ensure that all events sent to an object are received by the object, all scenarios are handled, and all events are handled that can affect an object in each of its states. The following checks are made for completeness and consistency at the component level:

- Sender and receiver for every event
- Input events from class to class match scenarios
- Event names are consistent
- All scenarios are handled
- All events which can affect the object, in any state, are handled

Control Scheme

During detailed design the control scheme reflected in the dynamic model must be translated into an implementation. This is necessary because, unlike for object classes, most object-oriented programming languages do not include constructs corresponding to states.

During design a control scheme is established to define the relationships among the system components. This scheme not only defines the interfaces between components, but also the timing requirements the components have to satisfy in response to an external event. There are three control schemes that might be used: sequential, concurrent, and event driven. In sequential or process driven control there is a single path of execution through the program. Control proceeds in order, or branches according to values within the application. If data is needed from another source, control is passed to the resource and then passed back to the requester along with the requested data. In concurrent control, two or more paths of execution may overlap in time, but each path proceeds in a sequential manner. The third scheme centralizes control in a control class (event manager) that passes events to other classes. If the class performing the processing needs additional input it returns control to the event manager. During preliminary design, internal and external control flows are examined and grouped to form one or more components. A control scheme is selected to implement each component based upon threads within the component, the interactions among the threads, and the control scheme and response requirements established during design. The three main control schemes are summarized as follows:

- Sequential control

  The sequential control scheme begins with an initial state and identifies the main control path(s) through a diagram that corresponds to the normally expected sequence(s) of events stimulated by an external event(s) received by the class. The names of the states encountered along this path are listed as a linear sequence. This becomes a sequence of statements in the program.

  Alternate paths that branch off the main path and rejoin it later are identified. These become conditional statements in the program. Backward paths that branch off the main path and rejoin it later are identified. These become loops in the program. If there are multiple backward paths that do not cross, they become nested loops in the program. Backward paths that do not nest can be implemented using GOTOs as a last resort.

  The states and transitions that are left correspond to exception conditions. They can be handled by techniques such as error subroutines, GOTOs, setting or testing of status flags, or exception handling supported by the language (e.g., ADA).

- Concurrent control

  Within the concurrent control scheme concurrent programming language tasks are initiated inside the class by a method of the class. They also are initiated from outside the class by an external control class.

  For classes that execute concurrently with other classes of the component but which do not contain concurrency themselves, the state diagram is translated as if for a sequential process. These classes are the tasks identified during preliminary design. The input and output events are examined to determine the priority, timing, and actions to occur in response to events (control may proceed without waiting for response to a request). The control pseudo code is adjusted to reflect the concurrent control requirements. A protocol is defined for the acknowledgment and transmission of input and output events, and incorporate the supporting control is incorporated into the pseudo code.

  If there is inherent concurrency within the class, the concurrent control paths within the state diagram are translated into separate tasks. If inter-task communication is required, examine the events coming in and out of the tasks to determine the priority, timing, and actions to occur in response to events. Adjust the control pseudo code to reflect the concurrent control requirements. A protocol is defined for the acknowledgment and transmission of input and output events, and the supporting control is incorporated into the pseudo code. Control is provided for the initiation, inter-task communications, and termination of tasks. If true concurrency cannot be supported by the tasks, modify the pseudo code to reflect sequential execution of the tasks.

- Event-driven control

  The event driven control scheme verifies that an appropriate class exists within the component and adds it if it doesn't. If the control class is to manage the state transitions of each class in the component, then the control class must possess information about the current state, the possible successor states, and the events which the class may receive.

  The state diagram is reviewed for all internal and external events. Those events that cause a transition from one state to another are identified. A control algorithm is identified for each state and represent the transition from a state as a message sent from the event manager. The transitions into and out of states are consolidated and represented in a table. The table includes any conditions that must be assessed if there is more than one possible state for a transition.

  The control class can be made responsible for event traffic between classes. If the threads of control within a class are viewed as a processing state of the class, then this approach could be considered to be similar to the previous approach where the event manger determines the next state for all classes. In this case, however, the threads of execution through an object instance are the responsibility of the class, and control is returned to the event manager when the thread of execution is completed or when the executing thread needs additional input. The state diagrams are translated as a sequential process.

  A control algorithm is defined for each thread of control through the class, representing the transition from all final states as a message sent to the event manager. A table for transitions into and out of the threads is defined. Any conditions on the transition that must be assessed to determine the next thread of execution are included. Also the implementation of the event manager which was started in preliminary design is included.

  If concurrent processing is needed, a consideration is made for directing inter-process communication through the event manager. Request handling can be made the responsibility of the event manager by requiring the event manager to retain the request until the receiving class notifies the event manager that it is prepared to receive and respond to a new request. Request handling can also be made the responsibility of the individual classes by designing message queues which the classes could check periodically or just prior to returning control to the event manager.

Validation is performed to ensure that control algorithms and the dynamic model correspond to the optimized detailed design model. This includes defining control mechanisms to ensure the integrity of derived entities, modifying algorithms to utilize the derived values, and representing separate algorithms for the calculation of derived values. Also, it is ensured that control algorithms and state transition diagrams reflect the events required to initiate new access operations as well as any modifications to the inheritance structures of the object model.

The signature represented in the algorithm is checked against the corresponding state transition diagram signature. It is also verified that the signature is correctly represented by all users of the operation.

Control methods are modified to increase efficiency. For example, costly functional methods are removed from loop structures where possible. The algorithm of the control method is modified to reflect the efficiencies. If necessary, associated data structures are modified

3.3.2.2.3 Detailed Functional Models

Developing the detailed design model requires the validation of the preliminary design details of the functional model, and the incorporation of additional functional detail. Note that functional models are developed, as applicable.

The purpose of the functional model during detailed design is to specify the implementation (method) for each operation for each class in the object model. The format of the functional model depends upon the complexity of the operations. For trivial operations such as obtaining the value of an attribute, the textual description of the operation may suffice. For complex operations, data flow diagrams may be developed, but they should not be considered a mandatory part of the functional model.

Additional functional detail during detailed design includes ensuring that there are sufficient operations to handle all requirements. Input and output data is identified from events. During detailed design, the methods (implementations) for operations are designed and a clear description of the operation in textual form is included. Data Flow Diagrams are developed to aid in understanding the method should the method be too complex to describe in textual terms. Often when leveling operations processes in data flow diagrams, it is decided that some of the sub operations are better placed on new classes which are then associated with the original class. Should this occur, the object model and dynamic model are adjusted to be consistent with the functional model.

The operations of each class are examined for ways to simplify, increase reuse, and encapsulate functionality. Suboperations are broken out, operations are moved up or down an inheritance hierarchy, or additional classes are created to encapsulate portions of an existing class' behavior. When this happens, then the object model and dynamic model are adjusted to be consistent with the functional model.

Signatures are completed from operations and sub operations. This includes all parameters and their data types, default values (if any), and constraints (if any). It also includes return types, and any default values and constraints.

Beginning with the functional descriptions developed during preliminary design, algorithms are defined for each computational process. This definition process starts with the tops of generalization hierarchies and works down to leaf classes. Verification of the signature, and logical intent of each operation is performed as it applies to different classes, and changes to the models are made where necessary. The algorithms are examined to determine possible error conditions and then extended to provide recovery. The need to extend signatures to accommodate the communication of error conditions is reviewed.

Verification is performed to ensure that the functional methods and the data flow diagrams correspond to the optimized detail design model. This includes defining algorithms to calculate derived or qualified entities, modifying algorithms to utilize the derived or qualified values rather

than calculating them, and representing a separate algorithm for the calculation of the derived value. Verification also includes ensuring that algorithms exist for all new access operations, that they are represented as operations on the object model, and that the functional model reflects any modifications to inheritance structures of the object model. The assignments of responsibility and the signatures of the operations are checked for consistency between the object model and any data flow diagrams. Also, the signature is verified as being correctly represented by all users of the operation. Existing functional methods are modified to increase efficiency. If necessary, the associated data structures are modified.

3.3.2.2.4 Segment Modeling

The segment diagrams depict the integration and interface of the components at a segment-level of abstraction. These diagrams may be either event flow diagrams or event trace diagrams. Their objective is to ensure that the lower level dynamic models and scenarios at the subsystem level properly integrate at the segment level to ensure full coverage and integration of the FOS.

3.3.2.2.5 Overall Consistency

The goal for detailed design is to produce a set of models that completely describe a solution to the problem and that are consistent with one another.

Models are internally consistent and consistent with one another. Detailed design is an iterative process in which the designer moves freely among the various models. As changes are made to one model, the other models are checked and modified, as necessary, to ensure continued consistency.

In order to ensure that the models are complete, L4 requirements are mapped to L3 requirements and L4 requirements are mapped to classes. This provides an unbroken chain by which high level requirements are broken down into requirements of ever increasing detail, while the corresponding design models are refined and mapped back to the requirements.

The partitioning of the object model is re-examined to validate the component boundaries in light of the completed definition of control and computational algorithms. Utility components are created and adjustments are made to existing application and utility component boundaries as appropriate.

The above inspections cannot be completed without a complete data dictionary. The data dictionary has entries for every modeling entity. It contains a clear description of each class, including the class owner, and any assumptions on its membership or use. The data dictionary also describes associations, attributes, and operations. A template is used, and the naming of entities is in accordance with the Software Development Plan.

3.4 FOS Development

Included in this section are the development naming conventions, the dynamic model template, the interface class description and FOS terms and concepts.

3.4.1 Naming Conventions

In the following examples 'Gr' represents the group (program, segment, or group) to which the code belongs. 'Li' represents the library or service. As stated in section 4.5.4 of the Software Development Plan (DID 308/DV2), the group and library acronyms are each two letters long and each start with a capital letter followed by one lower case letter.

Underscores are not allowed to precede the group acronym or separate the group and library acronyms. Underscores are allowed in MeaningfulName part of the name, **but their use is discouraged**.

The group acronyms for FOS are as follows:

Ec	ECS Common Code
Fo	FOS Common Code
Fa	Analysis
Fc	Commanding
Fd	Data Management
Fg	Real-Time Contact Management
Fm	Command Management
Fp	Planning and Scheduling
Fr	Resource Management
Ft	Telemetry
Fu	User Interface

The library/service names are created by each group. If a library gets too large, it may be divided into multiple libraries. There needs to be coordination on library names between the groups, but only for those libraries that are of a similar nature, such as Ut for Utilities. Here are some example library names:

Ut	Utilities
Tl	Telemetry
Cv	Current Value Table

3.4.1.1 Classes

All class names must start with the group and library in which they belong. This is followed by a descriptive name of the class. All class definitions will reside in include files. This convention will alleviate the problems of class collision across the program and with COTS library classes. The convention will also allow programmers unfamiliar with the code to immediately identify where a class is to be found. It is recommended that class member variables start with a lower case "**my"** followed by a descriptive name starting with an upper case letter. It is recommended that static member variables start with a lower case "**our"** followed by a descriptive name starting with an upper case letter. This allows a programmer to distinguish between member variables and local variables when inside member functions. Class member function names **do not contain** the group and library prefix. Class member function names start with a capital letter. This will allow for function overloading without confusing function names. The following is an example of a class, member variable, and member function name:

```
GrLiMeaningfulName
myMeaningfulName    // Member Variables
ourMeaningfulName   // Static Member Variables
```

MeaningfulName // Member Functions

3.4.1.2 Functions Names

All function names must start with the group and library in which they belong. This is followed by a descriptive name of the function. All functions will be prototyped and the prototypes will reside in include files. This convention will alleviate the problems of function collision across the program and with COTS library functions. The convention will also allow programmers unfamiliar with the code to immediately identify where a function is found. The following is an example of a function name:

GrLiMeaningfulName

In the case of a function that belongs to the common software (i.e. common across all segments), the first 15 characters of a function's name must be unique, this includes the group and library prefix. The requirement comes from compiler and linker restrictions from some of the supported hardware platforms.

3.4.1.3 Enumerated Type Variables

All enumerated type variable names must start with the group followed by a capital **E** followed by the library in which they belong. This is followed by a descriptive name of the enumerated type variable. All enumerated types will reside in include files. This convention will alleviate the problems of enumerated type variable collision across the program and with COTS library enumerated types. The convention will also allow programmers unfamiliar with the code to immediately identify where an enumerated type is to be found. The following is an example of an enumerated type variable name:

GrELiMeaningfulName

3.4.1.4 External Variables

All external variable names must start with the group followed by a capital **X** followed by the library in which they belong. This is followed by a descriptive name of the external variable. All external variables will reside in include files. This convention will alleviate the problems of external variable collision across the program and with COTS library external variables. The convention will also allow programmers unfamiliar with the code to immediately identify where an external variable is to be found. The following is an example of a external variable name:

GrXLiMeaningfulName

In the case of an external variable that belongs to the common software (i.e. common across all segments), the first 15 characters of an external variable's name must be unique, this includes the group and library prefix. The requirement comes from compiler and linker restrictions from some of the supported hardware platforms.

3.4.1.5 Const

All const variable names must start with the group followed by a capital **C** followed by the library in which they belong. This is followed by a descriptive name of the variable. All const variables will reside in include files. This convention will alleviate the problems of redefinition of const variables across the program and with COTS library const. The convention will also allow programmers unfamiliar with the code to immediately identify where a const variable is to be found. The following is an example of a const variable name:

GrCLiMeaningfulName

3.4.1.6 Number Define

All "#define" names must start with the group followed by a capital **D** followed by the library in which they belong. This is followed by a descriptive name of the variable. All "#defines" will reside in include files. This convention will alleviate the problems of redefinition of defines across the program and with COTS library defines. The convention will also allow programmers unfamiliar with the code to immediately identify where a "#define" is to be found. The use of "#defines" is strongly discouraged in 'C++', a const is recommended instead. The following is an example of a "#define" name:

GrDLiMeaningfulName

3.4.1.7 Macros

All macros must start with the group followed by a capital **M** followed by the library in which they belong. This is followed by a descriptive name of the macro. All macros will reside in include files. This convention will alleviate the problems of macro redefinition across the program and with COTS library macros. The convention will also allow programmers unfamiliar with the code to immediately identify where a macro is to be found. The use of macros is strongly discouraged in 'C++', an inline function is recommended instead. The following is an example of a macro name:

GrMLiMeaningfulName

3.4.1.8 Typedefs

All typedef names must start with the group followed by a capital **T** followed by the library in which they belong. This is followed by a descriptive name of the typedef. All typedefs will reside in include files. This convention will alleviate the problems of typedef collision across the program and with COTS library typedefs. The convention will also allow programmers unfamiliar with the code to immediately identify where a typedef is to be found. The following is an example of a typedef:

GrTLiMeaningfulName

3.4.1.9 Local Variables, Function Calling Parameters, Structure Members

Local variables, function calling parameters and structure members start with a lower case descriptive name. The reason behind this is that names starting with upper case have special meaning such as a const, #define, macro, typedef, extern, enum, function or a member function. The following is an example of a local variable name:

meaningfulName

3.4.1.0 Source and Include File Names

'C++' source files should end in '.cxx', and 'C++' header files should end in '.h'. The following is an example of 'C++' file names:

GrLiMeaningfulName.cxx

GrLiMeaningfulName.h

3.4.2 Dynamic Model Template

The following is a template of the dynamic model which is to be followed throughout FOS segment development:

x.y.3 <subsystem> Dynamic Model

<Summarize the set of scenarios that will be defined in the dynamic model for the subsystem.

For example:>

The following are the Planning and Scheduling subsystem scenarios, which are defined in this section.

Planning

Initial Scheduling

Final Scheduling

Late Changes

x.y.3.1 <scenario name> Scenario

x.y.3.1.1 <scenario name> Abstract

The scenario name should refer to the purpose of the scenario and should be descriptive of the scenario. It should not be named according to the stimulus, the product (unless the generation of the product is the purpose), nor should it be a synopsis of the processing.

The abstract should briefly summarize the scenario processing to set the stage -- should be 1-3 sentences. Note that x.y.3.1.3 provides the full description of the scenario. Also note that there can be more than 1 event trace diagram per scenario. This is a judgment call, but if several event traces are very similar (e.g., they have the same Summary Information) then they can be included in the same scenario.

x.y.3.1.2 <scenario name> Summary Information

Interfaces:

<Explanation: Provide a list of all of the subsystems interfaces participating in the scenario. It is assumed that the other subsystems participation in the scenario has been coordinated with the leaders of the other subsystems, however, it is always best to unambiguously document the common understandings. >

Stimulus:

<Explanation: The stimulus is the event which causes the initiation of the scenario. The description of the stimulus and the conditions under which it occurs should be very explicit.

The stimulus will appear on the event traces and the state diagrams, and will be evident in the detailed description of the scenario.

When the stimulus is externally generated then it will appear in the "Interface With Other Subsystems" section.>

Desired Response:

<Explanation: The desired response defines the successful completion of the scenario. The desired response need not be a single event. It may be a concurrent set of events such as the successful completion of updates to file#1, file#3, and the committing of updates to a database.

Specify the criteria which must be met to consider the response successful.

Where the desired response is an output to another subsystem or segment then it will appear in the "Interface With Other Subsystems" section.>

Participating Classes:

<Explanation: All classes defined in the object model(s) which participate in the scenario should be listed (and only listed -- i.e., no text). The details of the contents of the class will appear in the data dictionary.>

Pre-Conditions:

<Explanation: This section will provide any pre-conditions of the system, segment, and/or subsystem which affect the successful completion of the scenario. For example, the system will probably be required to be initialized via a successful cold start or warm restart, and some subset of the databases will probably need to be available and consistent, and ....>

The state of the system, segment and subsystem will affect the ability to successfully complete the scenario. An explicit exposition of the states will sensitize developers and reviewers to the criteria for success.>

Post-Conditions:

<Explanation: The state of the system, segment and subsystem may be affected by the state(s) which result from execution of a scenario. the ability to successfully complete the scenario.

This section should provide any post-conditions which result from the execution of the scenario.>

Example

Interfaces:

Data Management

User Interface

Stimulus:

Telemetry data sent from EDOS

Desired Response:

Decommutated, EU-converted, limit and delta checked telemetry parameters.

Participating Classes: <note: include just the object names; descriptions will be included in the data dictionary>

FTMFrame

FTMController

Pre-Conditions:

Application telemetry software has been initiated.

Telemetry data base run-time tables have been loaded.

Logical string defined and established to receive telemetry data from external source.

Post-Conditions:

Data dropout condition identified for telemetry data.

x.y.3.1.3 Scenario Description

The scenario description should expose as much as is known about the scenario at the current level of maturity. As much as possible the scenario should be a step by step traversal of the scenario.

The scenario should be accompanied by an event trace diagram(s) representing the scenario's path through the system or subsystem. The event trace diagram(s) should be explicitly referenced here. For example, Figure x.y.3.1.3-1.

x.y.3.1.4 State Transition Description

Where appropriate, provide the State Transition Diagrams for the subsystem and the objects under consideration. Include the narrative text describing the state transition diagram. Note: it may make more sense to include the state transition diagram and description in-line with the scenario description. Since we have not done this before, use your judgment ... please provide recommendations to the group, as applicable.

Wherever possible use the StP/OMT model components to document your work, and use the data dictionary to provide details.

x.y.3.2 <scenario name> Scenario

x.y.3.n <scenario name> Scenario

3.4.3 OMT Naming Conventions

Because C++ computer source code will be generated from the OMT tool, the naming conventions used must adhere to the ECS project naming conventions for interface classes. These conventions are spelled out in the ECS Project Instruction manual, under the C++ coding standard section.

In the following examples, *Gr* represents the group (program, segment, or group) to which the mode belongs, and *Li* represents the library or service.

The following naming conventions are to be used in the OMT tool:

ITEM	CONVENTION
Class/Object	*GrLi*MeaningfulName
Association	MeaningfulName
Attributes (member variables):	
member variables	myMeaningfulName
static member variables	ourMeaningfulName
enumerated types	meaningfulName
Operation (member function)	MeaningfulName
C and C++ Data Types	
const	*Gr*C*Li*MeaningfulName
enumerated types	*Gr*E*Li*MeaningfulName (external to a class)
#defines	*Gr*D*Li*MeaningfulName
typedefs	*Gr*T*Li*MeaningfulName
externs	*Gr*X*Li*MeaningfulName
Process (function)	*GrLi*MeaningfulName

3.4.4 FOS Terms and Concepts

The following list provides descriptions of terms used with reference to the FOS. This list is intended to aid in the clarification of these terms and to promote consistent usage of the terms within the segment.

1) In the L4 requirements, object models, etc. reference should be made to telemetry items as telemetry parameters as opposed to telemetry mnemonic or telemetry value. Telemetry mnemonic and telemetry values are attributes of a telemetry parameter.

2) In the L4 requirements, object models, etc. reference should be included to both subsystem and instrument. Subsystem pertains to the spacecraft subsystems (e.g., Command and Data Handling subsystem) while instrument pertains to MODIS, CERES, etc.

3) The term 'replay' should be used to refer to the replaying of historical data from the FOS archive, while the term 'playback' refers to the transfer of back-orbit telemetry from EDOS to the EOC. The term 'recorder dump' refers to the transfer of back-orbit telemetry from the spacecraft to EDOS.

4) The term 'derived parameter' should be used when referring to pseudo telemetry. Derived parameters may be data base defined, software defined, or user defined (as in procedures users can define). In the context of requirements, 'derived parameter' requirements should explicitly reference that these are data base defined since we currently do not have any software-defined derived parameters.

3.5 Configuration Management

The FOS will use the ECS Configuration Management Plan as a baseline for segment configuration management.

4. FOS Release Development Plan

This section contains two matrices representing the phases of development by identifying FOS scenarios and components. Both are intended to show a sequential flow of the development for each release. The first matrix is a summary scenario, which serves as a high level snapshot of the grouping of components within the FOS by release and build, and the activity phase in which these components are performed. The second matrix is a more detailed representation of the FOS scenario components utilized in the process of developing the segment. This second matrix includes the set of FOS scenarios, which shows the sequence and interdependencies of functional components within FOS scenarios that transcends individual FOS subsystems. This includes the relative sequencing in which the FOS will be developed.

These matrices will continue to be refined and are included for review. These matrices will be updated in the next release of the document.

Table 4-1. Scenario Summary Matrix (1 of 7)

Release	Build	Title	Description	Activity Phase	Key Functionality
A	1	Internal Connectivity	Communications level interfaces		
				Support	PDB Input
					Screen management(basic)
					Command language
					User authorization & authentication
					Event message processing(basic)
					Utilities (basic)
				Scheduling	Internal connectivity
					NCC connectivity
					Activity level constraints
					Receive and Validate Loads (basic)
				Real-Time	String configuration
					String connection(default)
					CCSDS packet processing

Table 4-1. Scenario Summary Matrix (2 of 7)

Release	Build	Title	Description	Activity Phase	Key Functionality
					Telemetry displays (data) pages
					Command Authorization
					Command entry and validation (basic)
				Analysis	Request preprocessing
	2	Basic telemetry and command	Provide telemetry and command paths, and end to end functionality		
				Support	PDB validation and ODB generation
					Data store/fetch
					DB maintenance activities (basic)
					Help
					Procedure Builder
					Events message processing (complete)
					Screen management (enhanced)
					Quick analysis tools
				Scheduling	Build BAP's (initial)
					Scheduling generation (s/c & inst.)
					Uplink load generation (basic)
					Receive and validate loads (enhanced)
					I/F connectivity -external
					Generate ground schedule
					Schedule deviations

Table 4-1. Scenario Summary Matrix (3 of 7)

Release	Build	Title	Description	Activity Phase	Key Functionality
				Real-Time	Command entry and validation (enhanced)
					Command generation
					Command/Load transmission(partial)
					Command authorization (enhanced)
					Command validation-Prerequisite check
					String connection(user)
					String configuration notification (tlm & cmd)
					Receive and store tlm data
					Display telemetry data (partial) plots, stripcharts, spreadsheets
					Telemetry processing (partial)
					Disseminate R/T data
					Multiple real-time contact monitoring
					NCC R/T interface
				Analysis	Analysis request processing
					Telemetry history processing(basic)
					Dataset generation
					Analysis Report generation
					Expert advisor
					Statistics generation (basic)
					Data Archive

Table 4-1. Scenario Summary Matrix (4 of 7)

Release	Build	Title	Description	Activity Phase	Key Functionality
B	1		Thread framework	Support	PDB validation and ODB generation (complete)
					Data base reporting (basic)
					Preplanned command procedure execution
					Advanced user interface functions
				Scheduling	FDF planning aids transfer
					I/F connectivity - external (complete)
					Activity and command level constraint check
					Build and validate load contents
					Schedule uplink loads
					Maintain uplink catalogs
					TDRS scheduling
					Display buffers and tables
				Real-Time	Command privilege change request processing
					Command request processing
					Ground script command processing
					Command receipt verification
					Command verification-telemetry

Table 4-1. Scenario Summary Matrix (5 of 7)

Release	Build	Title	Description	Activity Phase	Key Functionality
					Load validation and transmission (complete)
					Command reference processing (memory map, GRI)
					Telemetry processing (complete)
					Telemetry history processing (detailed)
					Ground telemetry processing (partial)
					Telemetry monitoring (partial)
					Display telemetry data (complete) - strip charts, graphs
					Replay processing(dedicated /shared)
					Microprocessor memory dump processing (partial)
					Spacecraft attitude data processing
					Merge telemetry data
				Analysis	Telemetry history processing
					Statistics generation (complete)
					State changes processing
					Report generation (customized)
					Standing Orders processing
					Clock correlation (partial)
B	2		Full FOS		

Table 4-1. Scenario Summary Matrix (6 of 7)

Release	Build	Title	Description	Activity Phase	Key Functionality
				Support	E-Mail
					User customization
					Data base reporting (complete)
					Data archive and retrieval
					Report browser/editor
					Screen management (complete)
					Data base maintenance activities (complete)
				Scheduling	Long term inst. plan receipt
					Generate timeline
					Update ground schedule
					Uplink load generation (complete)
					Schedule "what-if" plans
					Generate patch load
					Generate load reports
				Real-Time	Ground-tlm processing(complete)
					System Failure recovery
					Command configuration processing
					String reconfiguration
					Backup string configuration
					String termination
					Configuration monitoring (h/w, s/w)
					Telemetry monitoring (complete)

Table 4-1. Scenario Summary Matrix (7 of 7)

Release	Build	Title	Description	Activity Phase	Key Functionality
					Load verification
					Adjust telemetry parameters
					Downlink ordered report processing (complete)
					S/C state check processing
					Microprocessor memory dump processing (complete)
					Ground telemetry processing (complete)
				Analysis	Triggers processing
					Clock correlation (complete)
					Solid state recorder processing
					Apply special algorithms
					S/C activity log processing
Launch					

Table 4-2. FOS Scenario Matrix (1 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
Infrastructure - Comm	Communications	Interprocess Comm	CSS	A	1
		Multicast	CSS	A	1
		Security	ISS	A	1
		Name Service	CSS	A	1
		Network	ISS	A	1
		Comm I/F to SDPS	DMS	A	2
		Authentication	CSS	A	1
		Authorization	CSS	A	1
		Time	CSS	A	1
		Performance	MSS	B	1
		Planning	MSS	B	1
		CM	MSS	A	1
		IST Management	FOS	B	2
Support - DMS	Data Base	Tlm PDB Input	DMS	A	1
		Tlm PDB Validation	DMS	A	2
		Tlm DB Generation	DMS	A	2
		Maintain Tlm ODB Time Window	DMS	B	1
		Cmd PDB Input	DMS	A	1
		Cmd PDB Validation	DMS	A	2
		Cmd DB Generation	DMS	A	2
		Activity Input	DMS	A	1
		Activity Validation	DMS	A	2
		Activity Generation	DMS	A	2
		Constraint Input	DMS	B	1
		Constraint Validation	DMS	B	1
		Constraint Generation	DMS	B	1
		Data Base Reporting (Basic)	DMS	B	1
		Data Base Reporting (Detailed)	DMS	B	2
		Data Base Edit	DMS	A	2
		PDB Backup	DMS	B	2

Table 4-2. FOS Scenario Matrix (2 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
		PDB Restore	DMS	B	2
	File Management	Data Store	DMS	A	2
		Data Fetch	DMS	A	2
		Data CM	DMS	B	1
		Data Backup	DMS	B	1
		Data Restore	DMS	B	2
		Data Mover	FUI	B	2
		Display File Mgt	FUI	B	1
		Long-Term Archive to SDPS	DMS	B	2
		Long-Term Restore from SDPS	DMS	B	2
	Events	Events API and DB	DMS	A	1
		Events Msg Generation Basic	DMS	A	1
		Events Msg Generation Complete	DMS	A	2
		Display Events	FUI	A	1
		Quick Msg	FUI	B	2
		Archive Events	DMS	A	2
		Event History Request	FUI	B	1
		Retrieve Events	DMS	B	2
		Event History Display	FUI	B	2
Support - User Interface	General	User Authentication	CSS	A	1
		User Authentication Display	FUI	B	1
		Status Window	FUI	A	1
		Screen Management	FUI	A	1

Table 4-2. FOS Scenario Matrix (3 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
		Control Window Manip. (partial)	FUI	A	1
		Control Window Manip. (complete)	FUI	A	2
		User Customization	FUI	B	2
		Window Requirements/Snap	FUI	B	2
	Directive Input	Directive Language	FUI	A	1
		Procedure Builder	FUI	A	2
		Procedure Control	FUI	A	2
		Request Preplanned Command Procedure	FUI	A	2
		Generate Preplanned Command Procedure	FUI	A	2
		Validate Preplanned Command Procedure	CMS	B	2
		Preplanned Command Procedure Status	FUI	B	2
	Tools	Time Select- (partial)	FUI	A	1
		Time Select- (completel)	FUI	A	2
		Document Reader	FUI	B	2
		E-Mail	FUI	B	2
		Help - (partial)	FUI	A	2
		Help - (complete)	FUI	B	2
		Room Builder	FUI	A	1
		Display Builder - alpha	FUI	A	2
		Display Builder - alpha (conts.)	FUI	A	2
		Display Builder - graph/tables	FUI	B	1
		Display Builder - schematics	FUI	B	2
		Quick Analysis	FUI	A	2

Table 4-2. FOS Scenario Matrix (4 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
		Subsystem Filter	FUI	A	1
Scheduling	Long-Term Planning	Receive LTIP	PAS	B	2
		Receive LTSP	PAS	B	2
		Gen Long-Term S/C Op Plan	PAS	A	2
		Build/Define Activities in BAPs	PAS	A	2
		Gen Instrument BAPs	PAS	A	2
	Initial Scheduling	FDF Interface Connectivity	PAS	B	1
		Receive FDF Planning Aids	PAS	B	1
		Send FDF Planning Aids to SDPS	PAS	B	2
		Send FDF Planning Aids to ASTER	PAS	B	1
		Schedule Instrument BAPs	PAS	A	2
		Schedule Spacecraft Subsystem BAPs	PAS	B	1
		Schedule Spacecraft Subsystem Activities	PAS	A	2
		Generate Timeline	PAS	B	2
		Print Postscript Timeline	PAS	B	2
		ASTER Interface Connectivity	PAS	A	2
		Receive ASTER Instrument Activities	PAS	A	2
		Check Activity Level Constraints	PAS	B	1
		Schedule Instrument Activity Deviations	PAS	B	1

Table 4-2. FOS Scenario Matrix (5 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
		Schedule Spacecraft Subsystem Activity Deviations	PAS	B	1
		Receive ASTER Instrument Activity Deviations	PAS	B	1
		NCC Interface Connectivity (Basic)	PAS	A	1
		NCC Interface Connectivity (Complete)	PAS	B	1
		Create TDRSS Contact Schedule	PAS	B	1
		Submit TDRSS Contact Requests to NCC	PAS	B	1
		Receive Accepted TDRSS Contact Requests	PAS	B	1
		Receive Rejected TDRSS Contact Requests	PAS	B	1
		Reschedule Rejected TDRSS Contact Requests	PAS	B	2
	Final Scheduling	Schedule Instrument Activity Deviations	PAS	B	1
		Schedule Spacecraft Subsystem Activity Deviations	PAS	B	1
		Generate Timeline	PAS	B	2
		Print Postscript Timeline	PAS	B	2
		Receive ASTER Instrument Activity Deviations	PAS	B	1

Table 4-2. FOS Scenario Matrix (6 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
		Check Activity Level Constraints	PAS	B	1
		Schedule MP Load Uplink Activities	PAS	B	1
		Schedule Flight Software Uplink Activities	PAS	B	1
		Schedule RTS Uplink Activities	PAS	B	1
		Schedule Table Load Uplink Activities	PAS	B	1
	Daily Scheduling	Set Target Day (TD) Boundaries	PAS	A	2
		Freeze DAS Activities	PAS	A	2
		Eliminate Conflicting Activities	PAS	B	2
		Reschedule Deleted Activity	PAS	B	2
		Generate Timeline	PAS	B	2
		Print Postscript Timeline	PAS	B	2
		Expand Activity	CMS	A	2
		Check Command-level Constraints	CMS	B	1
		Build ATC Load for TD from DAS	CMS	A	2
		Generate ATC Uplink Load (Basic)	CMS	A	2
		Generate ATC Uplink Load (Complete)	CMS	B	2
		Determine Uplink Window	CMS	B	1
		Schedule ATC Uplink Activities	PAS	B	1

Table 4-2. FOS Scenario Matrix (7 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
		Add ATC Load Uplink to Ground Schedule	CMS	B	1
		Update Ground Schedule	CMS	A	2
		Generate Integrated Report	CMS	B	2
	Late Changes I (Load generated - not uplinked)	Receive Late Change Request	PAS	B	1
		Check Activity Level Constraints	PAS	B	1
		Eliminate Conflicting Activities	PAS	B	2
		Generate Timeline	PAS	B	2
		Print Postscript Timeline	PAS	B	2
		Expand Activity	CMS	A	2
		Check Command-level Constraints	CMS	B	1
		Build ATC Load for TD from DAS	CMS	A	2
		Generate ATC Uplink Load (Basic)	CMS	A	2
		Generate ATC Uplink Load (Complete)	CMS	B	2
		Determine Uplink Window	CMS	A	2
		Schedule ATC Uplink Activities	PAS	B	1
		Add ATC Load Uplink to Ground Schedule	CMS	B	1

Table 4-2. FOS Scenario Matrix (8 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
		Add ATC stored command verification to ground schedule	CMS	B	2
		Update Ground Schedule	CMS	A	2
		Generate Integrated Report		B	2
	Late Changes II (Load generated and uplinked) (patch)	Receive Late Change Request	PAS	B	2
		Check Activity Level Constraints	PAS	B	1
		Eliminate Conflicting Activities	PAS	B	2
		Generate Timeline	PAS	B	2
		Print Postscript Timeline	PAS	B	2
		Expand Activity	CMS	A	2
		Check Command-level Constraints	CMS	B	1
		Build ATC Partial Load from DAS	CMS	B	2
		Generate ATC Uplink Load (Basic)	CMS	B	2
		Generate ATC Uplink Load (Complete)	CMS	B	2
		Determine Uplink Window	CMS	B	2
		Schedule ATC Partial Load Uplink Activities	PAS	B	2

Table 4-2. FOS Scenario Matrix (9 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
		Add ATC Load Uplink to Ground Schedule	CMS	B	2
		Add ATC stored command verification to ground schedule	CMS	B	2
		Update Ground Schedule	CMS	B	2
		Generate Integrated Report	CMS	B	2
	What-If	Initiate What-If Plan	PAS	B	2
		Check Activity Level Constraints	PAS	B	1
		Expand Activity	CMS	A	2
		Check Command Level Constraints	CMS	B	1
		Schedule What-If	PAS	B	2
	Load Management	Receive Instrument Microprocessor Load Contents from SCF	FUI	A	2
		Validate MP Load Contents ID	FUI	A	2
		Generate Microprocessor Uplink Load	CMS	A	2
		Maintain Microprocessor Catalog	CMS	B	1
		Receive Flight Software Load Contents from SDVF	FUI	B	1

Table 4-2. FOS Scenario Matrix (10 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
		Validate Flight Software Load Contents ID	FUI	B	1
		Generate Flight Software Uplink Load	CMS	B	1
		Maintain Flight Software Catalog	CMS	B	1
		Receive Relative Time Sequence Load Contents	FUI	A	2
		Validate RTS Load Contents ID	FUI	A	2
		Load Manager	FUI	A	2
		Build RTS Load Contents	FUI	A	2
		Validate RTS Load Content	CMS	B	1
		Generate RTS Uplink Load	CMS	A	2
		Maintain RTS Uplink Load Catalog	CMS	B	1
		Receive Table Load Contents	FUI	A	2
		Validate Table Load Contents ID	FUI	A	2
		Build Table Load Contents	FUI	A	2
		Validate Table Load Contents	CMS	B	1
		Generate Table Uplink Load	CMS	A	2
		Generate FDF Table Loads	CMS	B	2
		Maintain Table Uplink Load Catalog	CMS	B	1
		Display ATC Buffer	FUI	B	1
		Display RTS Buffer	FUI	B	1

Table 4-2. FOS Scenario Matrix (11 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
		Generate Load Reports	CMS	B	2
Real-Time Operations	Real-Time System Initialization	Rqst Start-up Configuration Table	RMS	A	1
		Create Default Strings	RMS	A	1
		User Connection	RMS	A	1
	String Initialization	Request for new String	FUI	A	2
		User Requested String Creation	RMS	A	2
		Notify FUI of Configuration Change	RMS	A	2
		Configure CMD	RMS	A	1
		Configure TLM	RMS	A	1
		Configure RCM	RMS	A	1
	String Reconfiguration	String Reconfiguration Request	FUI	B	2
		Update String Configuration	RMS	B	2
		Notify FUI of Configuration Change	RMS	B	2
		Notify CMD of Configuration Change	RMS	B	2
		Notify TLM of Configuration Change	RMS	B	2
	String Termination	String Termination	RMS	B	2

Table 4-2. FOS Scenario Matrix (12 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
		Transfer Mission Critical Activity	RMS	B	2
		Termination of RTS	RMS	B	2
	Configuration Monitoring	Monitor Hardware	RMS	B	2
		Monitor Software	RMS	B	2
		Display Hardware/Software Status (partial)	FUI	A	2
		Display Hardware/Software Status (complete)	FUI	B	1
	Connection Support	User Authorization & Authentication	CSMS	A	1
		Real-Time Monitoring Request	FUI	A	1
		String Creation	RMS	A	1
		String Connection	RMS	A	1
	Managing Command Privilege	CMD Privilege Change Request	FUI	A	1
		Update CMD Privilege ACL	RMS	A	1
		CMD Authority	CMD	A	1
	Managing Configuration Privilege	Config Privilege Change Request	FUI	B	2
		Update ConFig Privilege ACL	RMS	B	2
Spacecraft Commanding	Command Authorization	Command Authorization Request	FUI	A	1

Table 4-2. FOS Scenario Matrix (13 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
		Command Authorization Approval	RMS	A	1
		Command Authorization Denial	RMS	A	1
		Notify CMD of new Commander	RMS	A	1
	Command Configuration	Command Configuration Request	FUI	B	1
		Command Configuration Modification	RMS	B	2
		Command Modes - FOP Configuration	CMD	B	2
		Command Modes - Prereq Check Configuration	CMD	B	2
	Ground Script Commanding	Ground Script Generate Request	FUI	A	2
		Generate ground script	CMS	A	2
		Display Historical Ground Script	FUI	B	2
		Ground Script Control (partial)	FUI	A	2
		Ground Script Control (complete)	FUI	B	1
		Command Processing	FUI	A	1
		Command Validation - DB Lookup	CMD	A	1

Table 4-2. FOS Scenario Matrix (14 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
		Command Validation - Prereq State Check	CMD	A	2
		Command Criticality	CMD	A	2
		Command Generation (Build CCSDS packet)	CMD	A	2
		Command Transmission to EDOS	CMD	A	2
		Command Verification - Receipt Verification	CMD	B	1
		Command Verification - Telemetry Verification	CMD	B	2
	Manual Commanding	Operator Command Entry	FUI	A	1
		Command Processing	FUI	A	1
		Accept CMD Directive	CMD	A	1
		Command Validation - DB Lookup	CMD	A	1
		Command Validation - Prereq State Check	CMD	A	2
		Command Criticality	CMD	A	2
		Command Generation (Build CCSDS packet)	CMD	A	2
		EBnet Interface Connectivity	ISS	A	1

Table 4-2. FOS Scenario Matrix (15 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
		Command Transmission to EDOS	CMD	A	2
		Command Verification - Receipt Verification	CMD	B	1
		Command Verfication - Telemetry Verification	CMD	B	2
		ASTER Command Notification	CMD	B	2
	Command Requests	Command Request Entry	FUI	B	2
		Command Request Submission	FUI	B	2
		EOC Notification	FUI	B	2
		Command Request Evaluation	FUI	B	2
		Command Request Accepted	FUI	B	2
		Originator Notification	FUI	B	2
		Command Request Processing	FUI	B	2
	Stored Cmd Verification	Ground Script Control (partial)	FUI	A	2
		Ground Script Control (complete)	FUI	B	1
		Command Processing	FUI	A	1
		Command Validation - DB Lookup	CMD	A	1

Table 4-2. FOS Scenario Matrix (16 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
		Command Verification - Telemetry Verification	CMD	B	2
	Load Processing	Operator Command Request Entry	FUI	B	1
		Command Processing	FUI	A	1
		Load Command Validation	CMD	B	1
		Load Criticality	CMD	B	2
		Load Transmission	CMD	B	1
		Load Verification	CMD	B	2
Telemetry	Real-Time Monitoring	String Connection Request	FUI	A	1
		String Connection	RMS	A	1
		Display Status Window	FUI	A	1
		Tlm Proc - CCSDS Packet Processing	TLM	A	1
		Tlm Proc - Decom	TLM	A	2
		Tlm Proc - Decom Health & Safety	TLM	A	2
		Tlm - Decom Housekeeping	TLM	A	2
		Tlm - Decom Houskeeping (cont.)	TLM	A	2
		Tlm Proc - EU Conversions	TLM	A	2
		Tlm Proc - Static	TLM	A	2
		Tlm Proc - Quality Flags	TLM	B	1
		Tlm Proc - Context Switches	TLM	B	1

Table 4-2. FOS Scenario Matrix (17 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
		Tlm Proc - Selective Decom	TLM	B	2
		Tlm Data Dropouts - Event	TLM	A	2
		Tlm Data Dropouts - Static	TLM	A	2
		Archive Telemetry	TLM	A	2
		Dump Processing	TLM	B	1
		Tlm Proc - Quality Flags	TLM	B	1
		Processing Derived Parameters	TLM	B	2
	EDOS CODAs Real Time	Receive CODAs	RCM	B	1
		Store CODAs	RCM	B	1
		Process CODAs	RCM	B	1
		Generate CODA Statistics	ANA	B	2
		Adjusting Limit Values	TLM	B	2
		Adjust EU parameters	TLM	B	2
		Selecting Limit Set (context switch)	TLM	B	1
		Selecting Limit Set (user)	TLM	B	2
		Parameter Server	TLM	A	1
		Displaying Telemetry Data	FUI	A	1
		Displaying Telemetry Plots	FUI	A	2
		Displaying Telemetry Plots (conts.)	FUI	A	2

Table 4-2. FOS Scenario Matrix (18 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
		Displaying Telemetry Strip Charts	FUI	B	1
		Displaying Spreadsheets	FUI	A	2
		Displaying Telemetry Schematics	FUI	B	1
	NCC Real Time	NCC Communications Test Message	RCM	A	2
		NCC GCMRS	RCM	A	2
		ACK/NACK Processing	RCM	A	2
		NCC Performance Data	RCM	B	1
		Archive NCC Data	DMS	B	1
		DSN ODMS	RCM	B	2
	Multiple Real-Time Contacts Monitoring	String Connection Request	FUI	A	2
		Display Telemetry Information Window	FUI	A	2
		String Connections	RMS	A	1
		Processing Multiple Telemetry Streams	RMS	A	1
		Storing Multiple Telemetry Streams	TLM	B	1
		Displaying Multiple Source Telemetry	FUI	B	2
	Monitoring a Dedicated Replay	Replay Controller	FUI	B	1
		Adjusting Replay Period	DMS	B	2

Table 4-2. FOS Scenario Matrix (19 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
		Adjusting Replay Rate	DMS	B	2
		Providing Telemetry Config Information	RMS	B	1
		Loading Selected Telemetry Data Base	TLM	B	1
		Request Replay Data	RMS	B	1
		Telemetry Configuration Request	FUI	A	2
		Adjusting Parameter Selection	TLM	B	2
		Adjusting Limit Values	TLM	B	2
		Adjust EU parameters	TLM	B	2
		Retrieving History Data (Basic)	DMS	A	2
		Metering History Data	DMS	B	1
		Processing History Telemetry	TLM	A	2
		Processing Derived Parameters	TLM	B	2
		Analyzing History Telemetry	ANA	B	2
		Displaying Replay Telemetry Pages	FUI	A	1
		Displaying Replay Telemetry Plots	FUI	A	2
		Displaying Replay Telemetry Strip Charts	FUI	B	1
		Displaying Replay Telemetry Schematics	FUI	B	1

Table 4-2. FOS Scenario Matrix (20 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
	Configuring Telemetry Processing	Determining Current/Default Tlm Config	RMS	A	1
		Providing Telemetry Config Information	RMS	A	1
		Displaying Config Information	FUI	A	1
		Loading Selected Telemetry Data Base	TLM	A	2
		Configuring Mode (mirrored)	TLM	A	2
		Configuring Mode (tailored)	TLM	B	2
		Selecting Telemetry Stream	TLM	A	1
		Telemetry Configuration Request	FUI	A	2
		Telemetry Configuration Modification	RMS	B	2
		Adjusting Parameter Selection	TLM	B	2
		Adjusting Limit Values	TLM	B	2
		Adjust EU parameters	TLM	B	2
		Selecting Limit Set (context switch)	TLM	B	1
		Selecting Limit Set (user)	TLM	B	2
		Adjusting Derived Param Processing Rate	TLM	B	2

Table 4-2. FOS Scenario Matrix (21 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
		Enabling/Disabling Telemetry Archiving	TLM	B	2
	System Generated statistics for Ground-Telemetry Data (NCC, EDOS)	Receive and Store Ground Telemetry (GT) Data	RCM	B	2
		Provide GT Data	RCM	B	2
		Compute and update GT statistical data	ANA	B	2
		Store GT Statistical Data	DMS	B	2
	Collecting S/C Attitude Information	Attitude Data Subset Request	FUI	B	1
		Collecting Attitude Data for FDF	TLM	B	1
		Send Attitude Data to FDF	TLM	B	1
		Ingest Playback	DMS	B	2
	Tlm Merge	Tlm Merge	DMS	B	2
		Archive Tlm	DMS	B	2
		Providing Telemetry Config Information	RMS	B	1
		Loading Selected Telemetry Data Base	TLM	B	1
		Request Replay Data	RMS	B	1
		Telemetry Configuration Request	RMS	B	1
		Adjusting Parameter Selection	TLM	B	2
		Adjusting Limit Values	TLM	B	2

Table 4-2. FOS Scenario Matrix (22 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
		Adjust EU parameters	TLM	B	2
		Retrieving History Data (Detailed)	DMS	B	1
		Metering History Data	DMS	B	1
		Processing History Telemetry	TLM	A	2
		Processing Derived Parameters	TLM	B	2
		Generate Tlm Statistics	ANA	B	2
Analysis	Tlm History	Build Analysis Request (History)	FUI	A	2
		Queue Manager	DMS	B	1
		Request Manager	ANA	A	2
		Generate DB ID	DMS	A	2
		Create Parameter List from History Request	ANA	A	2
		Providing Telemetry Config Information	RMS	B	1
		Loading Selected Telemetry Data Base	TLM	B	1
		Request Replay Data	RMS	B	1
		Telemetry Configuration Request	RMS	B	1
		Adjusting Parameter Selection	TLM	B	2
		Adjusting Limit Values	TLM	B	2
		Adjust EU parameters	TLM	B	2
		Retrieving History Data (Basic)	DMS	A	2

Table 4-2. FOS Scenario Matrix (23 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
		Processing History Telemetry	TLM	A	2
		Processing Derived Parameters	TLM	B	2
		Build Dataset	ANA	A	2
		Generate Carry-Out	ANA	A	2
		Apply Request to Dataset	FUI	A	2
		Display or Print History Data	FUI	B	1
		Store Request (optional)	DMS	A	2
	Historical Request Which crosses Data Base Boundaries	Build History Request	FUI	A	2
		Queue Manager	DMS	B	1
		Request Manager	ANA	A	2
		Generate DB Ids	DMS	A	2
		Partition request	ANA	B	1
		Build first set of Parameter List	ANA	B	1
		Providing Telemetry Config Information	RMS	B	1
		Loading Selected Telemetry Data Base	TLM	B	1
		Request Replay Data	RMS	B	1
		Telemetry Configuration Request	RMS	B	1
		Adjusting Parameter Selection	TLM	B	2
		Adjusting Limit Values	TLM	B	2
		Adjust EU parameters	TLM	B	2

Table 4-2. FOS Scenario Matrix (24 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
		Retrieving History Data (Basic)	DMS	A	2
		Processing History Telemetry	TLM	A	2
		Processing Derived Parameters	TLM	B	2
		Build Dataset	ANA	A	2
		Generate Carry-Out	ANA	A	2
		Build additional Parameter List	ANA	B	1
		Compute and Update Statistical data	ANA	A	2
	System Generated Statistics for FDF Data	Receive and store FDF Data	DMS	B	1
		Compute and update FDF statistical data	ANA	B	1
		Store FDF Statistics	DMS	B	1
	User Selected Statistics	Prepare Telemetry Statistics Request	FUI	A	2
		Queue Manager	DMS	B	1
		Request Manager	ANA	A	2
		Generate DB Id	DMS	A	2
		Build Parameter List from request	ANA	A	2
		Providing Telemetry Config Information	RMS	B	1
		Loading Selected Telemetry Data Base	TLM	B	1
		Request Replay Data	RMS	B	1
		Telemetry Configuration Request	RMS	B	1
		Adjusting Parameter Selection	TLM	B	2

Table 4-2. FOS Scenario Matrix (25 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
		Adjusting Limit Values	TLM	B	2
		Adjust EU parameters	TLM	B	2
		Retrieving History Data (Basic)	DMS	A	2
		Processing History Telemetry	TLM	A	2
		Processing Derived Parameters	TLM	B	2
		Compute and Update Statistical Data	ANA	A	2
	Algorithm Processing	Receive Algorithm information from user	FUI	B	2
		Designer Algorithm	ANA	B	2
		Prepare Request for Algorithm Processing	FUI	B	2
		Queue Manager	DMS	B	1
		Request Manager	ANA	A	2
		Providing Telemetry Config Information	RMS	B	1
		Loading Selected Telemetry Data Base	TLM	B	1
		Request Replay Data	RMS	B	1
		Telemetry Configuration Request	RMS	B	1
		Adjusting Parameter Selection	TLM	B	2
		Adjusting Limit Values	TLM	B	2
		Adjust EU parameters	TLM	B	2
		Retrieving History Data (Basic)	DMS	A	2

Table 4-2. FOS Scenario Matrix (26 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
		Processing History Telemetry	TLM	A	2
		Processing Derived Parameters	TLM	B	2
		Store Statistical Data	DMS	A	2
		Apply algorithm	ANA	B	1
		Build dataset	ANA	A	2
		Generate Carry-Out	ANA	A	2
		Display or Print Data	FUI	B	2
	State Change Statistics	State Changes Generation	ANA	B	1
		State Changes Archive	ANA	B	1
		Request State Change Reports	FUI	B	1
		State Changes Dataset Gen	ANA	B	1
		State Changes Report Gen	ANA	B	1
		Display State Changes Report	FUI	B	1
	Time Ordered Downlink Report	Prepare Downlink Report Request	ANA	A	2
		Queue Manager	DMS	B	1
		Request Manager	ANA	A	2
		Generate DB Id	DMS	A	2
		Providing Telemetry Config Information	RMS	B	1
		Loading Selected Telemetry Data Base	TLM	B	1
		Request Replay Data	RMS	B	1
		Telemetry Configuration Request	RMS	B	1

Table 4-2. FOS Scenario Matrix (27 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
		Adjusting Parameter Selection	TLM	B	2
		Adjusting Limit Values	TLM	B	2
		Adjust EU parameters	TLM	B	2
		Retrieving History Data (Basic)	DMS	A	2
		Processing History Telemetry	TLM	A	2
		Processing Derived Parameters	TLM	B	2
		Build Dataset (Time Ordered)	ANA	A	2
		Generate Carry-Out	ANA	A	2
		Generate Report	ANA	A	2
		Display or Print report	FUI	A	2
	Parameters-out-of-Limits Report	Prepare Parameters-out-of-limits request	FUI	A	2
		Provide out-of-limits data from DMS	ANA	B	2
		Generate out-of-limits Report	ANA	A	2
		Display or Print Report	FUI	A	2
	Customized User Reports	Report Template Builder	FUI	B	1
		Queue Manager	DMS	B	1
		Generated DB ID	DMS	A	2
		Tlm Dataset Preparation (build dataset)	ANA	A	2
		Generate Carry-Out	ANA	A	2

Table 4-2. FOS Scenario Matrix (28 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
		Report Generation (partial)	FUI	A	1
		Report Generation (complete)	FUI	A	2
		Report Browser/Editor	FUI	B	2
	Standing Orders	Request Standing Order	FUI	B	1
		Review Standing Order	FUI	B	1
		Initiate Standing Order	FUI	B	1
		Queue Manager	DMS	B	1
		Generate DBID	DMS	A	2
		Tlm Dataset Preparation (build dataset)	ANA	A	2
		Generate Carry-Out	ANA	A	2
		Display/Print Results	FUI	B	1
	Triggers	Initiate Trigger	DMS	B	2
		Queue Manager	DMS	B	1
		Generate DBId	DMS	A	2
		Tlm Dataset Preparation (build dataset)	ANA	A	2
		Generate Carry-Out	ANA	A	2
		Display/Print Trigger Results	FUI	B	2
	S/C Activity Log	Create Parameter List	ANA	A	2
		Tlm Proc-Decom	TLM	A	2
		S/C Activity Log Monitor	ANA	B	2

Table 4-2. FOS Scenario Matrix (29 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
		Display Status Window	FUI	A	1
	Expert Advisor	Tlm Proc-Decom	TLM	A	2
		RTworks Integration	ANA	A	2
		Displaying Telemetry Data	FUI	A	1
Segment	System Failure Recovery	Component Failure Detection	RMS	B	2
		Component Failure Recovery Rqst	FUI	B	2
		RTS H/W Failure Recovery	RMS	B	2
		DMS H/W Failure Recovery	RMS	B	2
		User Station H/W Failure Recovery	RMS	B	2
		RMS S/W Failure Recovery	RMS	B	2
		TLM S/W Failure Recovery	RMS	B	2
		CMD S/W Failure Recovery	RMS	B	2
		RCM S/W Failure Recovery	RMS	B	2
	Checking Spacecraft State	State Check Request	FUI	B	2
		Providing State Table	CMS	B	2
		Processing Telemetry	TLM	A	2
		Comparing Expected State with Telemetry	TLM	B	2
		Displaying Compare Results	FUI	B	2

Table 4-2. FOS Scenario Matrix (30 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
		State Table Baseline Request	FUI	B	2
		Baselining State Table	TLM	B	2
	Clock Correlation	Clock Correlation Request	PAS	B	1
		Processing TTMs	RCM	B	2
		Processing RCTDs	RCM	B	2
		Processing History Telemetry	TLM	B	1
		Calculating Clock Error	ANA	B	1
		Reporting Clock Error	ANA	B	1
		Displaying Clock Error	FUI	B	1
		Clock Error Report	ANA	B	1
	Memory Management	Maintain Ground Reference Image	CMS	B	1
		Export Memory Image Files	FUI, CMS	B	2
		Update Memory-to-Command Map	CMS	B	1
		Update Ground Reference Image	CMS	B	2
	Ingesting Memory Dump (S/C or instrument)	Microprocessor Memory Dump Request	FUI	B	1
		Configuring Memory Dump Processing	RMS	B	1
		Processing Memory Dump Telemetry	TLM	B	1

Table 4-2. FOS Scenario Matrix (31 of 31)

Activity Phase	Thread	Component	Subsystem	Release	Release A/B Build
		Build Dump Image from Colletected Tlm	CMS	B	2
		Memory Dump Compare	CMS	B	2
		Display/Print Compare Results	FUI	B	1
	Tlm Archive	Tlm Retrieve	DMS	B	1
		Ground Tlm Archive	DMS	A	2
		Ground Tlm Retrieve	DMS	B	2
	Solid State Recorder	SSR Analysis Window	FUI	B	2
		SSR Scheduling	PAS	B	1
		Receive CODA's	RCM	B	1
		Receive NCC Data	RCM	B	1
		Correlate SSR Data Scheduling	ANA	B	2
		SSR Recommendations	ANA	B	2
		SSR Real-Time Recommendations	ANA	B	2
		Provide SSR data to PAS	ANA	B	2

5. FOS Development Schedules

5.1 FOS Supplementary Master Schedule

The FOS supplementary master schedule serves as the baseline reference for the segment software turnover dates for releases A and B. Releases A and B functionality is developed in two builds for each release. The first build for each release is identified as an internal build. The second build for release A is considered an incremental delivery, and the second build for release B is final delivery. This schedule identifies milestones for FOS internal and external dependencies, design walkthroughs, code walkthroughs, and unit tests. Additionally, the turnover dates are relative to the thread, build and release levels. Copies of the FOS supplementary master schedule are available upon request from FOS project control.

This page intentionally left blank.

Abbreviations and Acronyms

ACL	Access Control List
AD	Acceptance Check/TC Data
AGS	ASTER Ground System
AM	Morning (ante meridiem) -- see EOS AM
Ao	Availability
APID	Application Identifier
ARAM	Automated Reliability/Availability/Maintainability
ASTER	Advanced Spaceborne Thermal Emission and Reflection Radiometer (formerly ITIR)
ATC	Absolute Time Command
BAP	Baseline Activity Profile
BC	Bypass check/Control Commands
BD	Bypass check/TC Data (Expedited Service)
BDU	Bus Data Unit
bps	bits per second
CAC	Command Activity Controller
CCB	Change Control Board
CCSDS	Consultative Committee for Space Data Systems
CCTI	Control Center Technology Interchange
CD-ROM	Compact Disk-Read Only Memory
CDR	Critical Design Review
CDRL	Contract Data Requirements List
CERES	Clouds and Earth's Radiant Energy System
CI	Configuration item
CIL	Critical Items List
CLCW	Command Link Control Words
CLTU	Command Link Transmission Unit
CMD	Command subsystem
CMS	Command Management Subsystem

CODA	Customer Operations Data Accounting
COP	Command Operations Procedure
COTS	Commercial Off-The-Shelf
CPU	Central Processing Unit
CRC	Cyclic Redundancy Code
CSCI	Computer software configuration item
CSMS	Communications and Systems Management Segment
CSS	Communications Subsystem (CSMS)
CSTOL	Customer System Test and Operations Language
CTIU	Command and Telemetry Interface Unit (AM-1)
DAAC	Distributed Active Archive Center
DAR	Data Acquisition Request
DAS	Detailed Activity Schedule
DAT	Digital Audio Tape
DB	Data Base
DBA	Database Administrator
DBMS	Database Management System
DCE	Distributed Computing Environment
DCP	Default Configuration Procedure
DEC	Digital Equipment Corporation
DES	Data Encryption Standard
DFCD	Data Format Control Document
DID	Data Item Description
DMS	Data Management Subsystem
DOD	Digital Optical Data
DoD	Department of Defense
DS	Data Server
DSN	Deep Space Network
DSS	Decision Support System
e-mail	electronic mail
Ecom	EOS Communication

ECS	EOSDIS Core System
EDOS	EOS Data and Operations System
EDU	EDOS Data Unit
EGS	EOS Ground System
EOC	Earth Observation Center (Japan);
	EOS Operations Center (ECS)
EOD	Entering Orbital Day
EON	Entering Orbital Night
EOS	Earth Observing System
EOSDIS	EOS Data and Information System
EPS	Encapsulated Postscript
ESH	EDOS Service Header
ESN	EOSDIS Science Network
ETS	EOS Test System
EU	Engineering Unit
EUVE	Extreme Ultra Violet Explorer
FAS	FOS Analysis Subsystem
FAST	Fast Auroral Snapshot Explorer
FDDI	Fiber Distributed Data Interface
FDF	Flight Dynamics Facility
FDIR	Fault Detection and Isolation Recovery
FDM	FOS Data Management Subsystem
FMEA	Failure Modes, and Effects Analyses
FOP	Frame Operations Procedure
FORMATS	FDF Orbital and Mission Aids Transformation System
FOS	Flight Operations Segment
FOT	Flight Operations Team
FOV	Field-Of-View
FPS	Fast Packet Switch
FRM	FOS Resource Management Subsystem
FSE	FOT S/C Evolutions

FTL	FOS Telemetry Subsystem
FUI	FOS User Interface
GB	Gigabytes
GCM	Global Circulation Model
GCMR	Global Circulation Model Request
GIMTACS	GOES I-M Telemetry and Command System
GMT	Greenwich Mean Time
GN	Ground Network
GOES	Geostationary Operational Environmental Satellite
GSFC	Goddard Space Flight Center
GUI	Graphical User Interface
H&S	Health and Safety
H/K	Housekeeping
HST	Hubble Space Telescope
I/F	Interface
I/O	Input/Output
ICC	Instrument Control Center
ICD	Interface Control Document
ID	Identifier
IDB	Instrument Database
IDR	Incremental Design Review
IEEE	Institute of Electrical and Electronics Engineers
IOT	Instrument Operations Team
IP	International Partners
IP-ICC	International Partners-Instrument Control Center
IPs	International Partners
IRD	Interface requirements document
ISDN	Integrated Systems Digital Network
ISOLAN	Isolated Local Area Network
ISR	Input Schedule Request
IST	Instrument Support Terminal

	Instrument Support Toolkit
IST	Instrument Support Toolkit
IWG	Investigator Working Group
JPL	Jet Propulsion Laboratory
Kbps	Kilobits per second
LAN	Local Area Network
LaRC	Langley Research Center
LASP	Laboratory for Atmospheric Studies Project
LEO	Low Earth Orbit
LOS	Loss of Signal
LSM	Local System Manager
LTIP	Long-Term Instrument Plan
LTSP	Long-Term Science Plan
MAC	Medium Access Control;
	Message Authentication Code
MB	Megabytes
MBONE	Multicast Backbone
Mbps	Megabits per second
MDT	Mean Down Time
MIB	Management Information Base
MISR	Multi-angle Imaging Spectro-Radiometer
MMM	Minimum, Maximum, and Mean
MO&DSD	Mission Operations and Data Systems Directorate (GSFC Code 500)
MODIS	Moderate resolution Imaging Spectrometer
MOPITT	Measurements Of Pollution In The Troposphere
MSS	Management Subsystem
MTPE	Mission to Planet Earth
NASA	National Aeronautics and Space Administration
Nascom	NASA Communications Network
NASDA	National Space Development Agency (Japan)
NCAR	National Center for Atmospheric Research

NCC	Network Control Center
NEC	North Equator Crossing
NFS	Network File System
NOAA	National Oceanic and Atmospheric Administration
NSI	NASA Science Internet
NTT	Nippon Telephone and Telegraph
OASIS	Operations and Science Instrument Support
ODB	Operational Database
ODM	Operational Data Message
OMT	Object Model Technique
OO	Object Oriented
OOD	Object Oriented Design
OpLAN	Operational LAN
OSI	Open System Interconnect
PACS	Polar Acquisition and Command System
PAS	Planning and Scheduling
PDB	Project Data Base
PDF	Publisher's Display Format
PDR	Preliminary Design Review
PI	Principal Investigator
PI/TL	Principal Investigator/Team Leader
PID	Parameter ID
PIN	Password Identification Number
POLAR	Polar Plasma Laboratory
POP	Polar-Orbiting Platform
POSIX	Portable Operating System for Computing Environments
PSAT	Predicted Site Acquisition Table
PSTOL	PORTS System Test and Operation Language
Q/L	Quick Look
R/T	Real-Time
RAID	Redundant Array of Inexpensive Disks

RCM	Real-Time Contact Management
RDBMS	Relational Database Management System
RMA	Reliability, Maintainability, Availability
RMON	Remote Monitoring
RMS	Resource Management Subsystem
RPC	Remote Processing Computer
RTCS	Relative Time Command Sequence
RTS	Relative Time Sequence;
	Real-Time Server
S/C	Spacecraft
SAR	Schedule Add Requests
SCC	Spacecraft Controls Computer
SCF	Science Computing Facility
SCL	Spacecraft Command Language
SDF	Software Development Facility
SDPS	Science Data Processing Segment
SDVF	Software Development and Validation Facility
SEAS	Systems, Engineering, and Analysis Support
SEC	South Equator Crossing
SLAN	Support LAN
SMA	S-band Multiple Access
SMC	Service Management Center
SN	Space Network
SNMP	System Network Mgt Protocol
SQL	Structured Query Language
SSA	S-band Single Access
SSIM	Spacecraft Simulator
SSR	Solid State Recorder
STOL	System Test and Operations Language
T&C	Telemetry and Command
TAE	Transportable Applications Environment

TBD	To Be Determined
TBR	To Be Replaced/Resolved/Reviewed
TCP	Transmission Control Protocol
TD	Target Day
TDM	Time Division Multiplex
TDRS	Tracking and Data Relay Satellite
TDRSS	Tracking and Data Relay Satellite System
TIROS	Television Infrared Operational Satellite
TL	Team Leader
TLM	Telemetry subsystem
TMON	Telemetry Monitor
TOO	Target Of Opportunity
TOPEX	Topography Ocean Experiment
TPOCC	Transportable Payload Operations Control Center
TRMM	Tropical Rainfall Measuring Mission
TRUST	TDRSS Resource User Support Terminal
TSS	TDRSS Service Session
TSTOL	TRMM System Test and Operations Language
TW	Target Week
U.S.	United States
UAV	User Antenna View
UI	User Interface
UPS	User Planning System
US	User Station
UTC	Universal Time Code; Universal Time Coordinated
VAX	Virtual Extended Address
VMS	Virtual Memory System
W/S	Workstation
WAN	Wide Area Network
WOTS	Wallops Orbital Tracking Station

XTE X-Ray Timing Explorer

This page intentionally left blank.

Glossary

activity	A specified amount of scheduled work that has a defined start date, takes a specific amount of time to complete, and comprises definable tasks.
analysis	Technical or mathematical evaluation based on calculation, interpolation, or other analytical methods. Analysis involves the processing of accumulated data obtained from other verification methods.
attitude data	Data that represent spacecraft orientation and onboard pointing information. Attitude data includes: • Attitude sensor data used to determine the pointing of the spacecraft axes, calibration and alignment data, Euler angles or quaternions, rates and biases, and associated parameters. • Attitude generated onboard in quaternion or Euler angle form. • Refined and routine production data related to the accuracy or knowledge of the attitude.
availability	A measure of the degree to which an item is in an operable and committable state at the start of a "mission" (a requirement to perform its function) when the "mission" is called for an unknown (random) time. (Mathematically, operational availability is defined as the mean time between failures divided by the sum of the mean time between failures and the mean down time [before restoration of function].
availability (inherent) (A_i)	The probability that, when under stated conditions in an ideal support environment without consideration for preventive action, a system will operate satisfactorily at any time. The “ideal support environment” referred to, exists when the stipulated tools, parts, skilled work force manuals, support equipment and other support items required are available. Inherent availability excludes whatever ready time, preventive maintenance downtime, supply downtime and administrative downtime may require. A_i can be expressed by the following formula: Ai = MTBF / (MTBF + MTTR) Where: MTBF = Mean Time Between Failures MTTR = Mean Time To Repair

availability (operational) (A_O)

The probability that a system or equipment, when used under stated conditions in an actual operational environment, will operate satisfactorily when called upon. A_O can be expressed by the following formula:

$$A_O = MTBM / (MTBM + MDT + ST)$$

Where: MTBM = Mean Time Between Maintenance (either corrective or preventive)

MDT = Mean Maintenance Down Time where corrective, preventive administrative and logistics actions are all considered.

ST = Standby Time (or switch over time)

baseline activity profile

A schedule of activities for a target week corresponding to normal instrument operations constructed by integrating long term plans (i.e., LTSP, LTIP, and long term spacecraft operations plan).

build

An assemblage of threads to produce a gradual buildup of system capabilities.

calibration

The collection of data required to perform calibration of the instrument science data, instrument engineering data, and the spacecraft engineering data. It includes pre-flight calibration measurements, in-flight calibrator measurements, calibration equation coefficients derived from calibration software routines, and ground truth data that are to be used in the data calibration processing routine.

command

Instruction for action to be carried out by a space-based instrument or spacecraft.

command and data handling (C&DH)

The spacecraft command and data handling subsystem which conveys commands to the spacecraft and research instruments, collects and formats spacecraft and instrument data, generates time and frequency references for subsystems and instruments, and collects and distributes ancillary data.

command group

A logical set of one or more commands which are not stored onboard the spacecraft and instruments for delayed execution, but are executed immediately upon reaching their destination on board. For the U.S. spacecraft, from the perspective of the EOS Operations Center (EOC), a preplanned command group is preprocessed by, and stored at, the EOC in preparation for later uplink. A real-time command group is unplanned in the sense that it is not preprocessed and stored by the EOC.

detailed activity schedules	The schedule for a spacecraft and instruments which covers up to a10 day period and is generated/updated daily based on the instrument activity listing for each of the instruments on the respective spacecraft. For a spacecraft and instrument schedule the spacecraft subsystem activity specifications needed for routine spacecraft maintenance and/or for supporting instruments activities are incorporated in the detailed activity schedule.
direct broadcast	Continuous down-link transmission of selected real-time data over a broad area (non-specific users).
EOS Data and Operations System (EDOS) production data set	Data sets generated by EDOS using raw instrument or spacecraft packets with space-to-ground transmission artifacts removed, in time order, with duplicate data removed, and with quality/ accounting (Q/A) metadata appended. Time span, number of packets, or number of orbits encompassed in a single data set are specified by the recipient of the data. These data sets are equivalent to Level 0 data formatted with Q/A metadata. For EOS, the data sets are composed of: instrument science packets, instrument engineering packets, spacecraft housekeeping packets, or onboard ancillary packets with quality and accounting information from each individual packet and the data set itself and with essential formatting information for unambiguous identification and subsequent processing.
housekeeping data	The subset of engineering data required for mission and science operations. These include health and safety, ephemeris, and other required environmental parameters.
instrument	• A hardware system that collects scientific or operational data. • Hardware-integrated collection of one or more sensors contributing data of one type to an investigation. • An integrated collection of hardware containing one or more sensors and associated controls designed to produce data on/in an observational environment.
instrument activity deviation list	An instrument's activity deviations from an existing instrument activity list, used by the EOC for developing the detailed activity schedule.
instrument activity list	An instrument's list of activities that nominally covers seven days, used by the EOC for developing the detailed activity schedule.
instrument engineering data	All non-science data provided by the instrument.

instrument microprocessor memory loads	Storage of data into the contents of the memory of an instrument's microprocessor, if applicable. These loads could include microprocessor-stored tables, microprocessor-stored commands, or updates to microprocessor software.
instrument resource deviation list	An instrument's anticipated resource deviations from an existing resource profile, used by the EOC for establishing TDRSS contact times and building the preliminary resource schedule.
instrument resource profile	Anticipated resource needs for an instrument over a target week, used by the EOC for establishing TDRSS contact times and building the preliminary resource schedule.
instrument science data	Data produced by the science sensor(s) of an instrument, usually constituting the mission of that instrument.
long-term instrument plan (LTIP)	The plan generated by the instrument representative to the spacecraft's IWG with instrument-specific information to complement the LTSP. It is generated or updated approximately every six months and covers a period of up to approximately 5 years.
long-term science plan (LTSP)	The plan generated by the spacecraft's IWG containing guidelines, policy, and priorities for its spacecraft and instruments. The LTSP is generated or updated approximately every six months and covers a period of up to approximately five years.
long term spacecraft operations plan	Outlines anticipated spacecraft subsystem operations and maintenance, along with forecasted orbit maneuvers from the Flight Dynamics Facility, spanning a period of several months.
mean time between failure (MTBF)	The reliability result of the reciprocal of a failure rate that predicts the average number of hours that an item, assembly or piece part will operate within specific design parameters. (MTBF=1/(l) failure rate; (l) failure rate = # of failures/operating time.
mean time between maintenance (MTBM)	The average time between all maintenance including both corrective and preventive maintenance.
mean time to repair (MTTR)	The mean time required to perform corrective maintenance to restore a system/equipment to operate within design parameters.
object	Identifiable encapsulated entities providing one or more services that clients can request. Objects are created and destroyed as a result of object requests. Objects are identified by client via unique reference.

orbit data	Data that represent spacecraft locations. Orbit (or ephemeris) data include: Geodetic latitude, longitude and height above an adopted reference ellipsoid (or distance from the center of mass of the Earth); a corresponding statement about the accuracy of the position and the corresponding time of the position (including the time system); some accuracy requirements may be hundreds of meters while other may be a few centimeters.
playback data	Data that have been stored on-board the spacecraft for delayed transmission to the ground.
preliminary resource schedule	An initial integrated spacecraft schedule, derived from instrument and subsystem resource needs, that includes the network control center TDRSS contact times and nominally spans seven days.
preplanned stored command	A command issued to an instrument or subsystem to be executed at some later time. These commands will be collected and forwarded during an available uplink prior to execution.
principal investigator (PI)	An individual who is contracted to conduct a specific scientific investigation. (An instrument PI is the person designated by the EOS Program as ultimately responsible for the delivery and performance of standard products derived from an EOS instrument investigation.).
prototype	Prototypes are focused developments of some aspect of the system which may advance evolutionary change. Prototypes may be developed without anticipation of the resulting software being directly included in a formal release. Prototypes are developed on a faster time scale than the incremental and formal development track.
raw data	Data in their original packets, as received from the spacecraft and instruments, unprocessed by EDOS. • Level 0 – Raw instrument data at original resolution, time ordered, with duplicate packets removed. • Level 1A – Level 0 data, which may have been reformatted or transformed reversibly, located to a coordinate system, and packaged with needed ancillary and engineering data. • Level 1B – Radiometrically corrected and calibrated data in physical units at full instrument resolution as acquired. • Level 2 – Retrieved environmental variables (e.g., ocean wave height, soil moisture, ice concentration) at the same location and similar resolution as the Level 1 source data.

	• Level 3 – Data or retrieved environmental variables that have been spatially and/or temporally resampled (i.e., derived from Level 1 or Level 2 data products). Such resampling may include averaging and compositing. • Level 4 – Model output and/or variables derived from lower level data which are not directly measured by the instruments. For example, new variables based upon a time series of Level 2 or Level 3 data.
real-time data	Data that are acquired and transmitted immediately to the ground (as opposed to playback data). Delay is limited to the actual time required to transmit the data.
reconfiguration	A change in operational hardware, software, data bases or procedures brought about by a change in a system's objectives.
SCC-stored commands and tables	Commands and tables which are stored in the memory of the central onboard computer on the spacecraft. The execution of these commands or the result of loading these operational tables occurs sometime following their storage. The term "core-stored" applies only to the location where the items are stored on the spacecraft and instruments; core-stored commands or tables could be associated with the spacecraft or any of the instruments.
scenario	A description of the operation of the system in user's terminology including a description of the output response for a given set of input stimuli. Scenarios are used to define operations concepts.
segment	One of the three functional subdivisions of the ECS: CSMS – Communications and Systems Management Segment FOS – Flight Operations Segment SDPS – Science Data Processing Segment
sensor	A device which transmits an output signal in response to a physical input stimulus (such as radiance, sound, etc.). Science and engineering sensors are distinguished according to the stimuli to which they respond. • Sensor name: The name of the satellite sensor which was used to obtain that data.
spacecraft engineering data	The subset of engineering data from spacecraft sensor measurements and on-board computations.

spacecraft subsystems activity list	A spacecraft subsystem's list of activities that nominally covers seven days, used by the EOC for developing the detailed activity schedule.
spacecraft subsystems resource profile	Anticipated resource needs for a spacecraft subsystem over a target week, used by the EOC for establishing TDRSS contact times and building the preliminary resource schedule.
target of opportunity (TOO)	A TOO is a science event or phenomenon that cannot be fully predicted in advance, thus requiring timely system response or high-priority processing.
thread	A set of components (software, hardware, and data) and operational procedures that implement a function or set of functions.
thread, as used in some Systems Engineering documents	A set of components (software, hardware, and data) and operational procedures that implement a scenario, portion of a scenario, or multiple scenarios.
toolkits	Some user toolkits developed by the ECS contractor will be packaged and delivered on a schedule independent of ECS releases to facilitate science data processing software development and other development activities occurring in parallel with the ECS.

This page intentionally left blank.